Débruitage vidéo et applications

Thibaud Ehret

université
PARIS-SACLAY

Débruitage vidéo et applications

Video denoising and applications

Thèse de doctorat de l'Université Paris-Saclay

Ecole Doctorale de Mathématique Hadamard (EDMH) n° 574
Spécialité de doctorat : Mathématiques appliquées
Unité de recherche : Centre Borelli (ENS Paris-Saclay), UMR 9010 CNRS
Référent : Ecole normale supérieure de Paris-Saclay

Thèse présentée et soutenue à Cachan, le 4 Juin 2020, par

Thibaud EHRET

Composition du jury:

Agnès Desolneux Professeure, ENS Paris-Saclay	Présidente
Giacomo Boracchi Associate Professor, Politecnico di Milano	Rapporteur
Karen Egiazarian Professor, Tampere University	Rapporteur
Sylvain Paris Research scientist, Adobe research	Examinateur
Oliver Wang Senior research scientist, Adobe research	Examinateur
Jean-Michel Morel Professeur, ENS Paris-Saclay	Directeur
Pablo Arias ENS Paris-Saclay	Codirecteur
Gabriele Facciolo Professeur, ENS Paris-Saclay	Codirecteur

Thèse de doctorat

NNT: NNT2020UPASN018

Abstract

This thesis focuses on denoising and in particular video denoising. For a long time, it was not possible to envisage sophisticated video denoising. This is due to constraints linked with video processing. While it is possible for an image to require up to a minute of processing, or even processing it remotely, this is not possible for a video because of the large number of frames that makes a even a short video. Even though the problem is very similar to image denoising, video denoising requires specific methods. We show here that not only using the temporal information in video is very important but also that efficient implementations on GPU allow for real time video denoising without any major modification of the methods. We also show that the mechanisms developed can be used for other applications such as anomaly detection and forgery detection.

The first part of the thesis focuses on patch-based methods. We first study one of the most popular and successful video denoising method called VBM3D. It combines self-similarity, by looking for similar patches both spatially and temporally, as well as patch filtering. We then take a more in-depth look at the patch search step for these patch-based methods. In particular we show the importance of temporal patch search in video. A patch search that searches in the entire video allows for better denoising. We also propose a novel causal and recursive video denoising method called NL-Kalman that tries to bridge the gap between lighter temporal filtering methods and heavier non-causal self-similar methods. Finally, we also show that patch-based methods can be implemented efficiently of GPU allowing for real time video denoising without any loss in quality.

We investigate the new trend of machine learning methods for denoising and low-level processing in the second part. We first propose one of the first neural network architecture for video denoising that can compete with the state of the art. In particular we introduce temporal information into the network using self-similarity similarly to patch-based methods. We also show that deep learning offers new opportunities. In particular, it allows for denoising without knowing the noise model. Thanks to the proposed framework, videos that have been processed with an unknown processing pipeline can still be denoised. Finally, we take a look at RAW mosaicked data. Indeed demosaicking and denoising are usually the two first steps of any image processing pipeline. Some deep learning methods suggest solving the two problems at the same time. We review the current state of the art for image demosaicking and show that deep learning methods are much better both qualitatively and quantitatively than hand-crafted methods. We also show that such demosaicking methods can be trained without requiring ground truth by simply using pairs of mosaicked images of the same scene. This process allows for realistic joint demosaicking and denoising and even single burst fine-tuning.

The last part of this thesis apply mechanisms used for denoising methods to other applications. In particular, we first look at the problem of anomaly detection in images. We show that this problem can be reduced to detecting anomalies in residual images in which noise and anomalies prevail. We also look at other applications of patch matching such as forgery detection with the detection of copy-paste forgeries. We analyze the theory of the the famous *PatchMatch* method before presenting how it can be used for copy-paste detection. We also apply an *a contrario* patch matching method to increase the robustness to similar objects.

Résumé

Cette thèse se concentre sur le débruitage et en particulier le débruitage vidéo qui a pendant longtemps été ignoré. La raison principale est les contraintes liées au traitement de la vidéo. Pour une image, il est acceptable d'avoir besoin de plusieurs minutes de traitement, voire même de devoir le faire sur un serveur distant. Cela n'est pas acceptable pour une vidéo en raison de la taille des données. Bien que le débruitage d'image et de vidéo sont des problèmes très similaires, il est nécessaire d'avoir des outils spécifiques adaptés à la vidéo. Nous montrons ici que l'information temporelle est très importante pour le traitement de la vidéo mais aussi que des implémentations efficaces sur carte graphique permettent d'avoir des méthodes temps-réel sans avoir à les modifier. Nous montrons aussi que les outils utilisés pour le débruitage peuvent être utilisés pour d'autres applications comme la détection d'anomalies ou la détection de falsification.

La première partie de cette thèse est dédiée aux méthodes à patches. Nous étudions dans un premier temps VBM3D, l'une des méthodes les plus populaires. Elle combine l'auto-similarité, en cherchant des patches similaires à la fois spatialement et temporellement, et le filtrage des patches. Nous étudions en détail l'étape de recherche de patches des méthodes à patches. Nous montrons en particulier l'importance de la recherche temporelle de patches dans la vidéo. En effet, une recherche plus globale permet d'avoir de meilleures performances de débruitage. Nous proposons aussi une nouvelle méthode de débruitage, appelée NL-Kalman, qui est à la fois récursive et causale. Avec NL-Kalman, nous essayons de réduire l'écart entre les méthodes de type filtrage temporelle qui sont légères et les méthodes non-causales basées sur l'auto-similarité qui, bien que très efficaces, sont beaucoup plus coûteuses. Enfin, nous montrons que les méthodes à patches peuvent être implémentées efficacement sur carte graphique. Cela permet d'avoir un débruitage temps-réel sans compromis sur la qualité.

Dans une deuxième partie, nous étudions une nouvelle tendance pour le débruitage et les traitements bas-niveaux basée sur les méthodes d'apprentissage. Nous proposons tout d'abord l'une des premières architectures de réseaux de neurones pour le débruitage vidéo qui est compétitif avec l'état de l'art. Pour cela nous introduisons de l'information temporelle dans le réseau grâce à l'auto-similarité, comme dans les méthodes à patches. Nous montrons aussi que les méthodes basées sur l'apprentissage profond offrent de nouvelles opportunités. En particulier, il devient possible de débruiter sans connaître le modèle du bruit. Grâce à la méthode proposée, même les vidéos traitées par une chaîne de traitement inconnue peuvent être débruitées. Nous considérons aussi des données brutes encore mosaïquées. En effet, le démosaïquage et le débruitage sont, en général, les deux premières étapes d'une chaîne de traitement d'images. Certaines méthodes d'apprentissage proposent de résoudre les deux problèmes en même temps. Après avoir examiné l'état de l'art du démosaïquage, nous pouvons conclure que les méthodes basées sur les réseaux de neurones sont meilleures de façon quantitative mais aussi visuel que les autres méthodes. Nous montrons aussi que ces méthodes peuvent être entraînées sans vérité terrain en utilisant uniquement des paires d'images mosaïquées de la même scène. Ce procédé permet d'apprendre un démosaïquage et débruitage réaliste et même d'adapter l'apprentissage à une acquisition rafale spécifique.

La dernière partie de cette thèse concerne les applications à d'autres problèmes des mécanismes utilisés pour le débruitage. L'un de ces problèmes est la détection d'anomalies dans les images. Nous montrons que ce problème peut être ramené à détecter des anomalies dans une image résiduelle dans laquelle du bruit et les anomalies dominent. Nous étudions d'autres applications à la recherche de patches comme la détection de falsifications telles que la détection de copié-collé. Nous analysons la théorie derrière la célèbre méthode *PatchMatch* avant de présenter comment elle peut être utilisée pour la détection de copié-collé. Nous utilisons aussi la comparaison de patches *a contrario* pour augmenter la robustesse aux objets similaires.

Remerciements

Je tiens tout d'abord à remercier mes directeurs de thèse (Jean-Michel Morel, Pablo Arias et Gabriele Facciolo) pour l'opportunité ainsi que leur aide durant la préparation de cette thèse. Je remercie aussi mes rapporteurs (Giacomo Boracchi et Karen Egiazarian), mes examinateurs (Sylvain Paris et Oliver Wang) ainsi que la présidente de mon jury (Agnès Desolneux) pour leur participation à mon jury et cela malgré les conditions particulières de la soutenance.

Je remercie aussi Axel, Jérémy et Mauricio pour leur aide et participation à de nombreux projets, Roger, pour avoir survécu à mes tentatives de guide touristique, le SAV IPOL de mon bureau (Miguel), Guillermo pour ses nombreuses invitations pour venir travailler à Duke durant l'été ainsi que le reste des truffes de la team Duke (Tristan, Thibaud, Charles, Sébastien et Marie), Tina pour ses nombreuses interruption dans le bureau ainsi que l'ensemble des doctorants présents en même temps que moi au CMLA. Je remercie aussi énormément l'équipe du secrétariat, Alina, Véronique et Virginie, pour leur travail de l'ombre. Grâce à leur aide j'ai pu me concentrer pleinement sur mon travail sans avoir à me soucier de nombreux problèmes administratifs. Pour finir, je remercie aussi ma famille et mes amis (même ceux qui m'ont demandé de ne pas les remercier ici) pour leur support durant ces années.

J'oublie certainement de nombreuses personnes, du laboratoire ou en dehors, qui ont influencées de près ou de loin cette thèse et je m'en excuse mais sachez que je vous remercie tout autant que les autres.

Contents

Introduction

Motivations

Denoising is a fundamental image and video processing problem. While the performance of denoising methods and imaging sensors has steadily improved over decades of research, new challenges have also appeared. High-end cameras still acquire noisy images in low lighting conditions. High-speed video cameras use short exposure times, reducing the SNR of the captured frames. Cheaper, lower quality sensors are used extensively, for example in mobile phones or surveillance cameras, and require denoising even with a good scene illumination. This state of facts is amplified by the poor quality of the optic used for these devices, due to cost and space constraints.

Its relevance has increased with the democratization of mobile imaging devices such as smartphones due to the wide variety of usage. For example, these small devices are often used in poor lighting condition. This degrades the quality of the output image, and the dominant factor in that quality loss is noise. An example of the necessity of denoising even for high-end modern smartphones is shown in Figure 1.

Figure 1: While the final output of a Samsung Galaxy S7 doesn't show apparent noise (left), the RAW image seen by the sensor is actually very noisy (right). The final output is obtained thanks to a complex processing that applies denoising as one of its step.

The same problems also arise with other image sensing devices. In particular the same trend of smaller and cheaper sensors acquiring many images quickly can be observed in remote sensing imaging. For example, one of the largest provider of satellite images *Planet* can now acquire short videos, also called bursts. While these images have a worst ground resolution and more noise, multi-frame denoising and super-resolution [MSS+14] can be applied to create a single higher-quality image. Denoising has also been applied to non-optical imaging devices such as

radar images [DDT+14], ultrasound [CHKB09], fMRI images [WR04] and fluorescent images [BKB+09] for medical images.

An additional problem with modern devices is the prevalence of proprietary pipelines. Images are available at the end of an usually unknown processing pipeline or, sometimes, completely raw data. The noise at the end of the pipeline is difficult to model and quite different from the popular Gaussian noise model assumed in many methods. While the noise of raw data is easy to model, a difficulty is caused by mosaicked data. Indeed, most cameras capture only one color per pixel, this color being determined by a color filter array (CFA) located on top of the sensor. Such data requires additional processing steps. In this thesis we try to take into account all practical challenges that come with modern denoising by looking at the possibilities offered by model-blind denoising and also looking at joint demosaicking and denoising.

While denoising is the main focus of this thesis, we also take a look at the problem of anomaly detection. The automatic detection of anomalous structure in arbitrary images is concerned with the problem of delineating regions not conforming with the rest of the data. For example, this is particularly important to detect defects in a production line. We shall put in evidence a striking similarity between methods for anomaly detection and for denoising. We also take a look at forgery detection, another problem with similarity to both denoising and anomaly detection.

Part I (Chapters 1 to 4) presents and extends traditional patch-based methods. In Part II (Chapters 5 to 8), we take a look at the more recent trend of machine learning for low-level vision processing and the new exciting opportunities they bring to the table. Finally, Part III (Chapters 9 to 12) deals with detection applications using similar tools than the ones developed for denoising.

1. VBM3D and its extensions

In this chapter we review one of the most popular video denoising methods. VBM3D is an extension to video of the well-known image denoising algorithm BM3D, which takes advantage of the sparse representation of stacks of similar patches in a transform domain. The extension is rather straightforward: the similar 2D patches are taken from a spatio-temporal neighborhood which includes neighboring frames. In spite of its simplicity, the algorithm offers a good trade-off between denoising performance and computational complexity. A detailed description is given and the choice of parameters is thoroughly discussed. Furthermore, we study and compare several extensions of the original algorithm: (1) a multi-scale implementation, (2) the use of 3D patches, (3) the use of optical flow to guide the patch search. These extensions allow to obtain results which are competitive with even the most recent state of the art. Figure 2 presents the core structure of VBM3D.

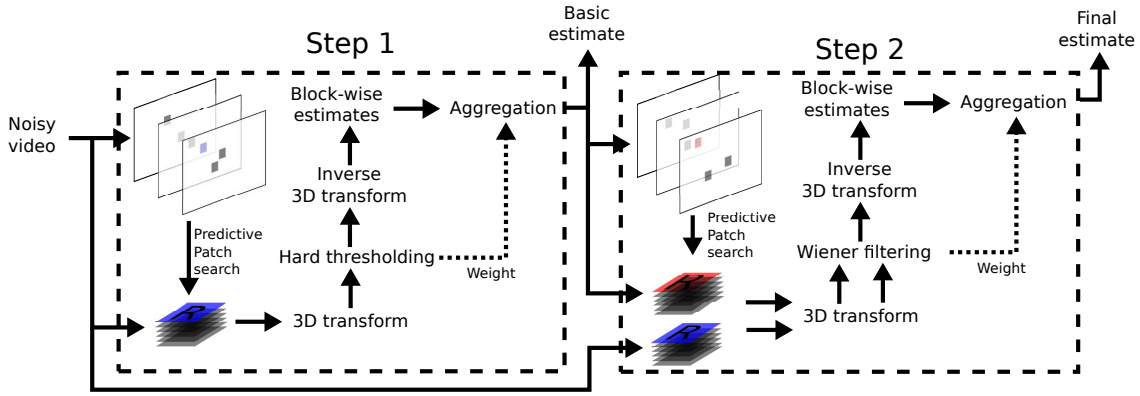


Figure 2: Scheme of the core of the VBM3D algorithm.

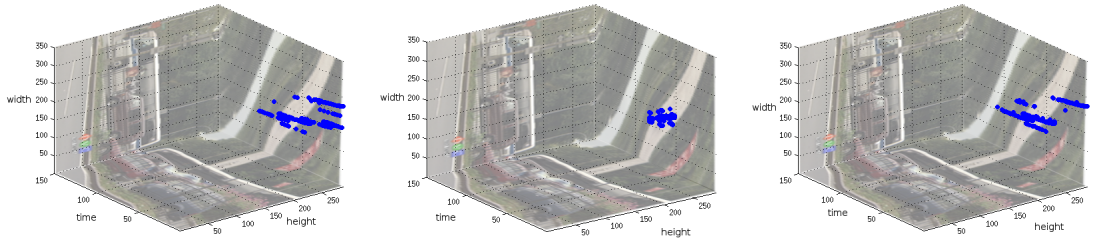


Figure 3: The plots show the position in the spatio-temporal video domain of the matches found for a sample patch query for different search methods. From left to right: the best matches found with a global exhaustive search, with a local exhaustive search in a window centered at the query, and with the VPLR search, the heuristic proposed in this chapter. Notice how the latter discovers the patch trajectories similar to those of the global exhaustive search.

This method will then be used as reference to compare the denoising methods proposed in the following chapters. Understanding in details all the mechanisms of the method also led to an optimized GPU implementation presented in Chapter 4.

2. Global patch-search for denoising

As we show in Chapter 1, patch search is an important mechanism of patch-based methods, and it will be recurrent throughout the thesis. In the case of a single image, a local search region is justified by the fact that similar patches are likely to be close to each other in the image domain. Videos however, have an additional strong source of redundancy given by the temporal consistency. A patch is expected to have similar patches along its motion trajectory, even in distant frames. It seems intuitive that patch-based methods should benefit from this larger set of similar exemplars. However, this search is generally performed locally around each target patch for obvious complexity reasons.

In this chapter we focus on the patch search. We present an efficient global approximate search technique and demonstrate its impact on video denoising. It permits for the first time to evaluate the impact of a global search on the video denoising performance.

Figure 3 shows that the method achieves a behavior similar to a global exhaustive search, for a fraction of the computational cost.

3. A recursive video denoising method: NL-Kalman

Contrary to the methods presented in Chapters 1 and 2, we propose a recursive causal method in this chapter. This means that the method uses only the current frame and the previous denoised one. The goal of such methods is to close the gap between high-quality but slow methods and those efficient but of lower quality.

The proposed method, called NL-Kalman, considers the video as a set of overlapping temporal patch trajectories. Following a Bayesian approach each trajectory is modeled as a linear dynamic Gaussian model and denoised by a Kalman filter. To estimate its parameters, similar patches are grouped and their trajectories are considered as sharing the same model parameters. The filtering is mainly temporal; non-local spatial similarity is only used to estimate the parameters. This temporally causal method obtains results comparable (in terms of PSNR and SSIM) to state-of-the-art methods using several frames per frame denoised but with a better temporal consistency. While temporal consistency cannot be visualized without looking at the video, Figure 4

Figure 4: Denoising results when the sequence is corrupted by noise of standard deviation 30. From left to right: Denoising using NL-Kalman, NL-Kalman with an oracle optical flow, VBM3D, VBM4D and the original frame. From top to bottom: Crop from the sequence `crowd_run` and `pedestrian_area`. Overall the proposed method preserves more details as it can be seen both in the tree and in the text. Results best viewed zoomed.

Method	SF	GPU	PSNR	Time/frame
NLM ours	yes	yes	31.54	0.851 ms
BM3D ours	yes	yes	32.93	4.51 ms
NLM video ours	no	yes	33.53	10.6 ms
VBM3D [EA20]	no	no	34.21	2.90 s
VBM3D ours	no	yes	34.31	3.30 ms

Table 1: Comparison of denoising performance (average PSNR) and average running time per frame of BM3D, VBM3D and NL-means on the Derf dataset of the compared implementations (noise standard deviation of level 20). Whether the method uses a single frame (SF) or a GPU is indicated.

shows that the proposed method also recovers more details.

4. Fast patch-based denoising on GPU

We show in the previous chapters that denoising is an essential part of any image or video processing pipeline. Unfortunately, due to time processing constraints, many pipelines do not consider the use of modern denoisers. These algorithms have only CPU implementations or suboptimal GPU implementations. In this chapter, we propose a new efficient GPU implementation of NL-means and BM3D, and, to the best of our knowledge, the first GPU implementation of the video denoising algorithm VBM3D. The performance of these implementations enable their use in real-time scenarios. Table 1 shows the impact of an optimized implementation for video denoising.

5. Non-Local video denoising by CNN

Non-local patch based methods were until recently state of the art for image denoising but are now outperformed by machine learning and in particular CNNs. Yet they are still the state of the art for video denoising, as video redundancy is a key factor to attain high denoising performance. The problem is that CNN architectures are hardly compatible with the search for self-similarities.

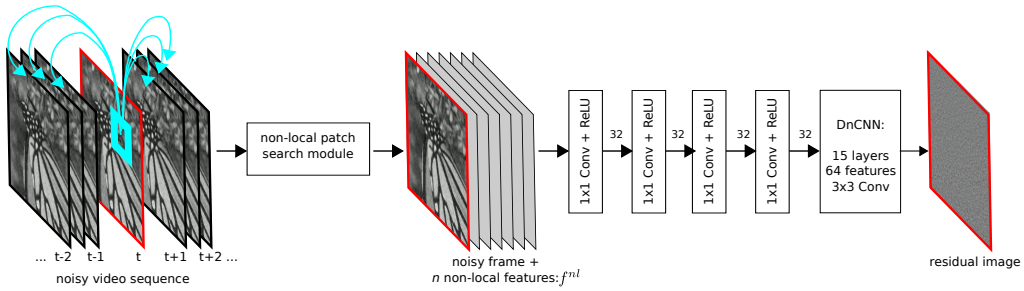


Figure 5: The architecture of the proposed method. The first module performs a patch-wise nearest neighbor search across neighboring frames. Then, the current frame, and the feature vectors f^{nl} of each pixel (the center pixels of the nearest neighbors) are fed into the network. The first four layers of the network perform 1×1 convolutions with 32 feature maps. The resulting feature maps are the input of a simplified DnCNN [ZZC+17a] network with 15 layers.

In this chapter we propose a new and efficient way to feed video self-similarities to a CNN. The non-locality is incorporated into the network via a first non-trainable layer which finds for each patch in the input image its most similar patches in a search region. The central values of these patches are then gathered in a feature vector which is assigned to each image pixel. This information is presented to a CNN which is trained to predict the clean image. We apply the proposed architecture to image and video denoising. For video, patches are searched for in a 3D spatio-temporal volume. The proposed architecture, shown in Figure 5, achieves state-of-the-art results.

6. Model-blind video denoising via frame-to-frame training

In the previous chapters, the noise model is always considered fixed and known. However, modeling the processing chain that has produced a video is a difficult reverse engineering task, even when the camera is available. This makes model based video processing a still more complex task. In this chapter we propose a fully blind video denoising method, with two versions offline and online. This is achieved by fine-tuning a pre-trained AWGN denoising network to the video with a novel frame-to-frame training strategy. Our denoiser can be used without knowledge of the origin of the video and the post-processing steps applied from the camera sensor. The online process only requires a couple of frames before achieving visually pleasing results for a wide range of perturbations. It nonetheless reaches state-of-the-art performance for standard Gaussian noise, and can be used offline with still better performance. The flexibility of the proposed method is shown in Figure 6.

7. Demosaicking with deep learning

Most cameras capture the information of only one color for a given pixel. This results in a mosaicked image that must be interpolated to get three colors at each pixel. The step going from a mosaicked image to a regular RGB image is called demosaicking. This chapter reviews two recent demosaicking methods based on convolutional neural networks that achieve artifact-free state-of-the-art results: Deep joint demosaicking and denoising by Gharbi *et al.* [GCPD16] and Color image demosaicking via deep residual learning by Tan *et al.* [TZZZ17]. These methods beat by almost two decibels the best human-crafted methods, while being faster by one order of magnitude. A difficult example of demosaicking where only learning methods performs well is shown in Figure 7. This, arguably, seals the destiny of human-crafted methods on this subject.

Figure 6: From the *same* starting point and *only* using the video, our fine-tuned network is able to denoise different noises without any artifact. The top images are the noisy and the bottom ones the denoised. From left to right: Gaussian noise, Poisson type noise, salt and pepper type noise and JPEG compressed Gaussian noise.

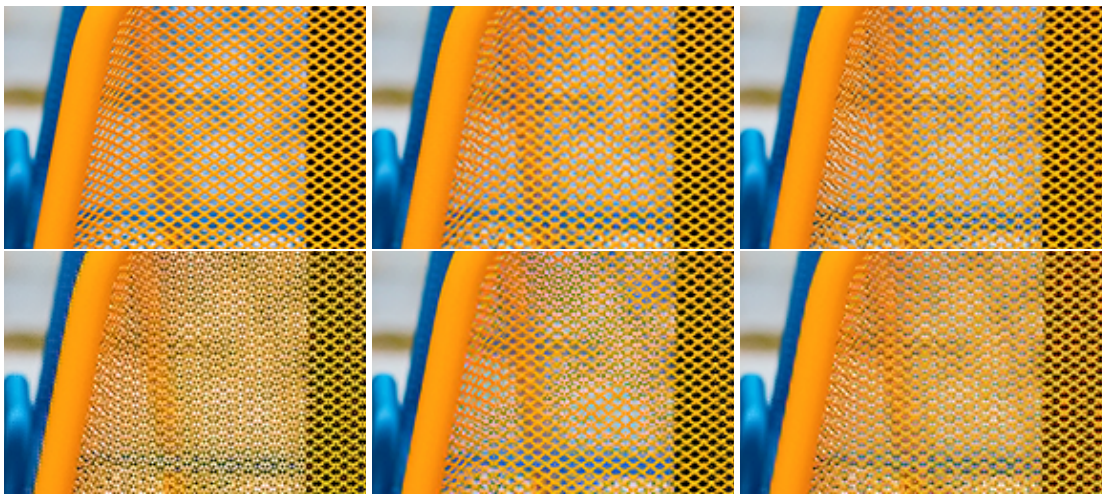


Figure 7: From top-left to bottom-right: Original, Gharbi *et al.* [GCPD16], Tan *et al.* [TZZZ17], MLRI [KMTO14], ARI [MKTO15], Getreuer [Get12]. Learning based methods produce fewer artifacts while still being very efficient computationally.

Figure 8: Using a burst, our fine-tuning (starting from the network from Gharbi *et al.* [GCPD16]) is able to not only denoise well ($\sigma = 5$) but also doesn't show any artifacts like *zipper* or *moire* in the difficult regions. Best visualized on a screen.

8. Joint demosaicking and denoising by fine-tuning of bursts of raw images

Demosaicking, studied in Chapter 7, and denoising are the first steps of any camera image processing pipeline and are key for obtaining high quality RGB images. A promising current research trend aims at solving these two problems jointly using convolutional neural networks. Due to the unavailability of ground truth data these networks cannot be currently trained using real RAW images. Instead, they resort to simulated data.

In this chapter we present a method to learn demosaicking directly from mosaicked images, without requiring ground truth RGB data. We apply this to learn joint demosaicking and denoising only from RAW images, thus enabling the use of real data. This allows to mitigate the training bias due to simulated data. In addition we show that for this application fine-tuning a network to a specific burst improves the quality of restoration for both demosaicking and denoising. An example on synthetic data is shown in Figure 8.

9. How to Reduce Anomaly Detection in Images to Anomaly Detection in Noise

While denoising is important to improve the visual quality of images and videos, it can also have a surprising application to seemingly completely unrelated problems. One of such surprising applications is detecting anomalies. Anomaly detectors address the difficult problem of detecting automatically exceptions in a background image, that can be as diverse as a fabric or a mammography. Detection methods have been proposed by the thousands because each problem requires a different background model. By analyzing the existing approaches, we show in this chapter that the problem can be reduced to detecting anomalies in residual images (extracted from the target image) in which noise and anomalies prevail. Hence, the general and impossible background modeling problem is replaced by a simple noise model and allows the calculation of rigorous detection thresholds. Our approach is therefore unsupervised and works on arbitrary images. The residual images can favorably be computed on dense features of neural networks. Our detector is powered by the *a contrario* detection theory, which avoids over-detection by fixing detection thresholds taking into account the multiple tests. Figure 9 shows detections on real natural images using different methods for anomaly detection.

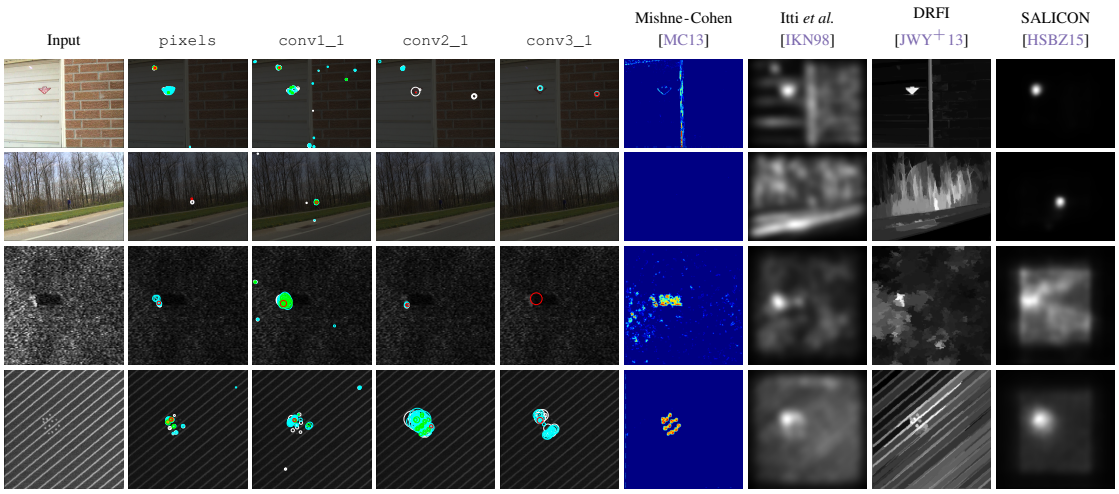


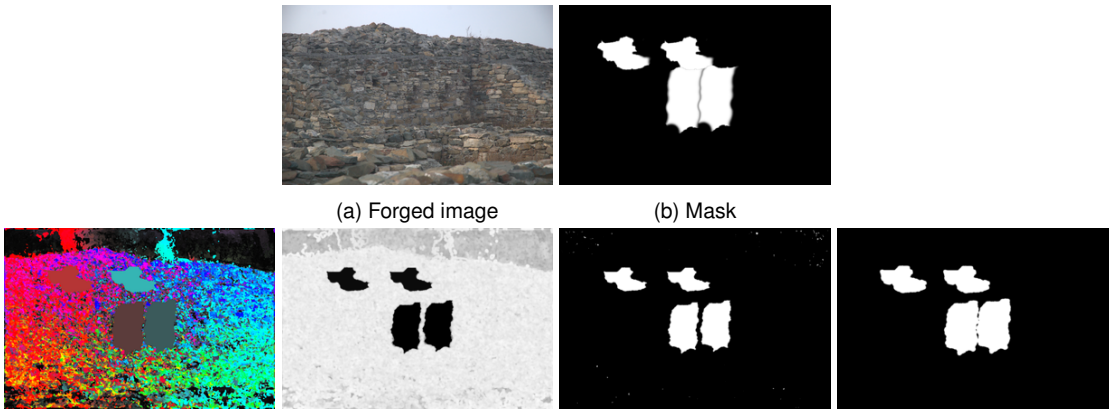
Figure 9: Detection results of our method when using as image representation directly the image pixels or the activation maps of the VGG neural Network at different layers (conv1_1, conv2_1, conv3_1) and a comparison to [MC13], [IKN98], [JWY+13] and [HSBZ15] on real examples. The first two images (top and second row) are part of the Toronto dataset [BT06], while the third and fourth row are from [MC13] and [TH99] respectively.

10. Analysis of PatchMatch

As seen in the previous chapters, many problems in image/video processing and computer vision require the computation of a dense k-nearest neighbor field (k-NNF) between two images. For each patch in a query image, the k-NNF determines the positions of the k most similar patches in a database image. With the introduction of the *PatchMatch* algorithm [BSFG09a], Barnes *et al.* demonstrated that this large search problem can be approximated efficiently by collaborative search methods that exploit the local coherency of image patches. After its introduction, several variants of the original *PatchMatch* algorithm have been proposed, some of them reducing the computational time by two orders of magnitude. In this chapter we study the convergence of *PatchMatch* and its variants, and derive bounds on their convergence rate. We consider a generic *PatchMatch* algorithm from which most specific instances found in the literature can be derived as particular cases. We also derive more specific bounds for two of these particular cases: the original *PatchMatch* and Coherency Sensitive Hashing [KA11]. The proposed bounds are validated by contrasting them to the convergence observed in practice.

11. Detection of copy-paste forgeries based on PatchMatch

This chapter reviews an application of the PatchMach process studied in Chapter 10. The method studied in this chapter has been presented in [CPV15]. This method is a forgery detection based on a dense field of descriptors chosen to be invariant by rotation, for example Zernike moments. An efficient matching of the descriptors is then performed using PatchMatch, which is extremely efficient to find duplicate regions. Regions matched by PatchMatch are processed to find the final detections. This allows a precise and accurate detection of copy-move forgeries inside a single suspicious image. We also extend successfully the method to the use of dense SIFT descriptors and show that they are better at detecting forgeries using Poisson editing. An example of detection is shown in Figure 10.



(a) Forged image (b) Mask

(c) Displacement map (d) Error map (e) Initial detection mask (f) Final detection mask

Figure 10: Detection of forgery using PatchMatch and Zernike moments for a forgery applying only a translation.

12. Detection of copy-paste forgeries based on sparse descriptors

The method presented in Chapter 11 suffers from over-detecting similar but genuinely different regions. This is because detecting reliably copy-move forgeries is difficult and in particular because images do contain similar objects. These similar objects are detected because most methods are not robust enough to differentiate clones and similar objects. The question then is: how to discard natural image self-similarities while still detecting copy-moved parts as being "unnaturally similar"? Especially knowing that copy-move may have been performed after a rotation, a change of scale and followed by JPEG compression or the addition of noise.

For this reason, we propose another method based on robust patch matching similar to those presented in Part I. We use keypoints similar to SIFT, meaning sparse keypoints with scale, rotation and illumination invariant descriptors. To discriminate natural descriptor matches from artificial ones, we introduce an *a contrario* patch comparison method which gives theoretical guarantees on the number of false alarms. We validate our method on several databases. Being fully unsupervised it can be integrated into any generic automated image tampering detection pipeline. In Figure 11, we show how it is possible to distinguish similar but genuine objects and forgeries using only local image patches.

13. Publications

This thesis has led to the following publications:

- **Global patch search boosts video denoising**, Thibaud Ehret, Pablo Arias and Jean-Michel Morel, *International Conference on Computer Vision Theory and Applications, 2017*
- **Automatic Detection of Internal Copy-Move Forgeries in Images**, Thibaud Ehret, *Image Processing On Line, 2018*
- **Model-blind video denoising via frame-to-frame training**, Thibaud Ehret, Axel Davy, Jean-Michel Morel, Gabriele Facciolo and Pablo Arias, *Conference on Computer Vision and Pattern Recognition, IEEE, 2018*
- **Non-Local Kalman: A recursive video denoising algorithm**, Thibaud Ehret, Jean-Michel Morel and Pablo Arias, *International Conference on Image Processing, IEEE, 2018*

Figure 11: On the left image the two objects are similar but not digital copies of each other. On the right, one is a digital copy of the other. The patches shown below each respective image correspond to the red squares in the images. They show that a difference is visible at this level and therefore the descriptors can be discriminated. This is why the detection method can discard genuinely similar objects.

- **On the convergence of PatchMatch and its variants**, Thibaud Ehret and Pablo Arias, *Conference on Computer Vision and Pattern Recognition, IEEE, 2018*
- **Reducing Anomaly Detection in Images to Detection in Noise**, Axel Davy, Thibaud Ehret, Jean-Michel Morel and Mauricio Delbracio, *International Conference on Image Processing, IEEE, 2018*
- **A Non-Local CNN For Video Denoising**, Axel Davy, Thibaud Ehret, Jean-Michel Morel, Pablo Arias and Gabriele Facciolo, *International Conference on Image Processing, IEEE, 2019*
- **Video Denoising by Combining Patch Search and CNNs.**, Axel Davy, Thibaud Ehret, Jean-Michel Morel, Pablo Arias and Gabriele Facciolo, *in Journal of Mathematical Imaging and Vision, Springer, 2021*
- **A Study of Two CNN Demosaicking Algorithms**, Thibaud Ehret and Gabriele Facciolo, *Image Processing On Line, 2019*
- **How to Reduce Anomaly Detection in Images to Anomaly Detection in Noise**, Thibaud Ehret, Axel Davy, Jean-Michel Morel and Mauricio Delbracio, *Image Processing On Line, 2019*
- **Image Anomalies: A Review and Synthesis of Detection Methods**, Thibaud Ehret, Axel Davy, Jean-Michel Morel and Mauricio Delbracio, *in Journal of Mathematical Imaging and Vision, Springer, 2019*
- **Joint Demosaicking and Denoising by Fine-Tuning of Bursts of Raw Images**, Thibaud Ehret, Axel Davy, Pablo Arias and Gabriele Facciolo, *International Conference on Computer Vision, IEEE, 2019*
- **GPU acceleration of NL-means, BM3D and VBM3D**, Axel Davy and Thibaud Ehret, *Journal of Real-Time Image Processing, Springer, 2020*

Introduction en français

Motivations

Le débruitage est un problème fondamental du traitement de l'image et de la vidéo. Bien que les performances des méthodes et des capteurs aient continué de s'améliorer au cours de dizaines d'années de recherche, de nouveaux défis sont apparus. Les appareils haut de gamme produisent encore des images bruitées dans des conditions de lumière difficiles. Les caméras ultra-rapide ont des temps de captures très courts, dégradant ainsi le rapport signal à bruit dans les images. Des capteurs moins chers et de moins bonne qualité sont utilisés à grande échelle pour les téléphones portable et les caméras de surveillance. Ces capteurs produisent des images bruitées même dans de bonnes conditions d'illumination. Ce problème est d'autant plus important que l'optique utilisée pour ces appareils est en général de mauvaise qualité à cause de leur coût et des contraintes de place.

Son importance est d'autant plus grande avec la démocratisation des téléphones portables à cause de leurs utilisations très variées. Par exemple, ces appareils sont souvent utilisés dans des environnements peu éclairés. Cela dégrade la qualité de l'image produite et le facteur prédominant de cette perte de qualité est le bruit. Un exemple de l'importance du débruitage même avec un téléphone haut de gamme récents est montré dans la Figure 12.

Figure 12: Bien que l'image produite par un téléphone Samsung Galaxy S7 ne semble pas être bruitée (gauche), l'image brute correspondante telle que vue par le capteur est en fait très bruitée (droite). L'image finale est produite par une chaîne complexe de traitement d'image qui applique un débruitage lors d'une des différentes étapes.

Les mêmes problèmes sont aussi présents avec d'autres systèmes d'acquisition d'images. En particulier, la tendance d'utiliser des capteurs plus petits et bon marché est aussi présente dans l'imagerie satellitaire. Par exemple, un des plus gros fournisseur d'images satellitaires, *Planet*,

peut maintenant acquérir de courtes séquences d'images. Bien que ces images aient une moins bonne résolution spatiale et plus de bruit, du débruitage et de la super-résolution multi-image permettent de créer une unique image de bien meilleure qualité. Le débruitage a aussi été utilisé pour d'autres types d'imagerie comme des images radar [DDT+14], ultrason [CHKB09], IRM [WR04] et de l'imagerie par fluorescence [BKB+09] pour des images médicales.

Un problème supplémentaire avec les appareils modernes est l'existence de chaînes de traitement propriétaires. Les images sont soient disponibles après une chaîne de traitement inconnue, soient, dans certains cas, complètement brutes. Le bruit après la chaîne de traitement est difficile à modéliser et peut être très différent d'un bruit blanc gaussien qui est très souvent le modèle supposé par les méthodes. Bien que le bruit des images brutes est plus facile à modéliser, leur problème vient du mosaïquage des données. En effet, la très grande majorité des appareils ne capture qu'une couleur par pixel, cette couleur étant déterminée par le filtre de couleur (CFA) situé sur le capteur. Ces données nécessitent du coup des traitements supplémentaires. Dans cette thèse, nous essayons de prendre en compte tous les défis pratiques du débruitage "moderne" en étudiant le débruitage dit "aveugle", sans avoir besoin de connaître le modèle de bruit à l'avance, et la résolution jointe du débruitage et du démosaïquage.

Bien que le débruitage soit le sujet principal de cette thèse, nous considérons aussi le problème de détection d'anomalies. La détection automatique de structures anomaliques dans des images arbitraires a pour but de délimiter les régions non conformes avec le reste des données. Par exemple, il est particulièrement important de détecter les défauts d'un produit dans une chaîne de production. Nous mettrons en lumière une ressemblance frappante entre la détection d'anomalies et le débruitage. Nous étudierons aussi la détection de falsification, un autre problème avec de grandes similarités avec le débruitage et la détection d'anomalies.

La Partie I (Chapitres 1 à 4) présente et étend les méthodes à patches pour le débruitage vidéo. Dans la Partie II (Chapitres 5 à 8), nous considérons le nouvel axe prometteur de recherche qui propose d'utiliser des méthodes d'apprentissage pour résoudre les problèmes de traitement d'images et vidéos et les nouvelles opportunités que ces méthodes offrent. Pour finir, la Partie III (Chapitres 9 à 12) présente différentes applications de détection qui utilisent des outils similaires à ceux présentés pour le débruitage.

1. VBM3D et ses extensions

Dans ce chapitre nous étudions l'une des méthodes les plus populaires pour le débruitage vidéo. VBM3D est une extension à la vidéo du célèbre algorithme de débruitage d'image BM3D qui utilise une représentation parcimonieuse des groupes de patches similaires après transformation. L'extension est assez directe: les patches 2D similaires sont trouvés dans un voisinage spatio-temporel qui inclut des images vidéo voisines. Malgré sa simplicité, l'algorithme offre un bon compromis entre qualité de débruitage et temps de calcul. Une description détaillée de la méthode est donnée et le choix des paramètres discuté. Nous étudions aussi différentes extensions possible à la méthode originale: (1) une implémentation multi-échelle, (2) l'utilisation de patches 3D, (3) l'utilisation du flot optique pour guider la recherche de patches. Ces extensions permettent d'obtenir des résultats compétitif avec l'état de l'art. La Figure 13 présente le principe de fonctionnement de VBM3D.

Cette méthode est utilisée ensuite comme référence pour comparer les différentes méthodes proposées dans les chapitres suivants. La bonne compréhension de tous les mécanismes de la méthode a permis d'implémenter efficacement la méthode sur carte graphique. Cette implémentation est présentée dans le Chapitre 4.

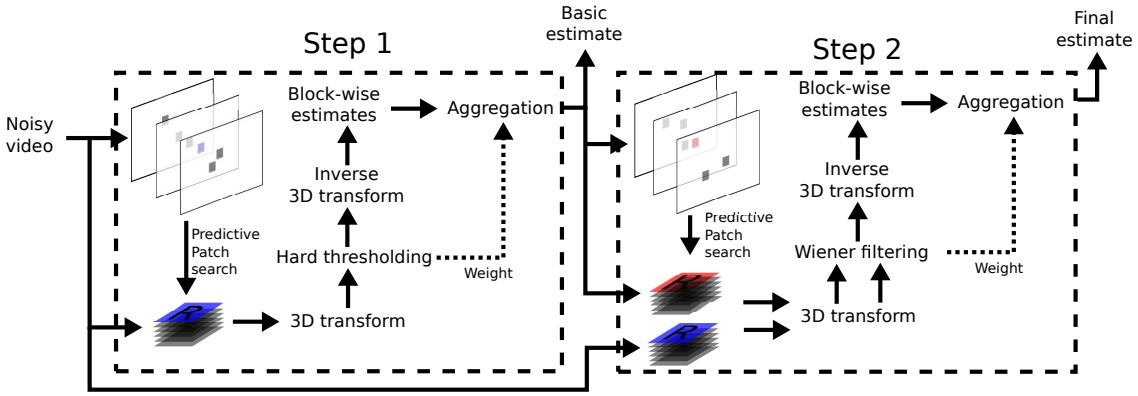


Figure 13: Schéma expliquant le principe de fonctionnement de VBM3D.

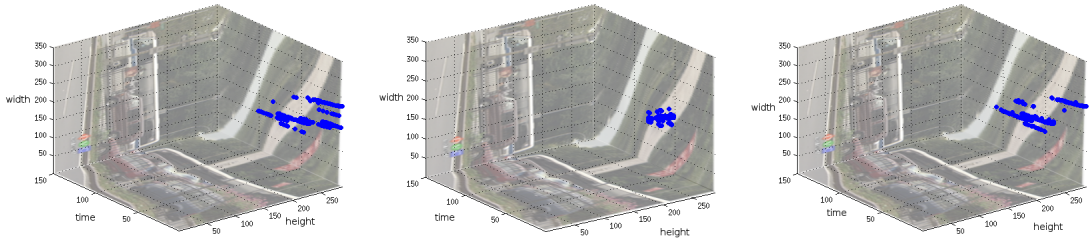


Figure 14: Ces images représentent la position des meilleurs patches dans la vidéo trouvés par différentes techniques de recherche de patches. De gauche à droite: la position des meilleures correspondances pour la recherche exhaustive globale, pour la recherche exhaustive locale centrée sur la position du patche de référence et l'heuristique proposée dans ce chapitre. Il est important de remarquer que cette dernière trouve des trajectoires de patches semblables aux trajectoires trouvées par la recherche exhaustive globale.

2. Recherche de patches globale pour le débruitage

Comme nous l'avons montré dans le Chapitre 1, la recherche de patches est un mécanisme important des méthodes à patches. Ce mécanisme est récurrent tout au long de cette thèse. Dans le cas d'une image unique, chercher dans une région locale est justifié par le fait que les patches les plus similaires sont plus susceptibles de se retrouver proche dans l'image. Les vidéos ont cependant une autre source de redondance grâce à la consistance temporelle. Il est normal de s'attendre à retrouver des patches similaires le long des trajectoires temporelles même dans des images vidéo distantes dans le temps. Il semble intuitif que les méthodes à patches peuvent bénéficier de cette source de similarité. Malheureusement, la recherche est cependant faite de façon locale pour chaque patche à cause de contraintes de temps de calcul évidentes.

Dans ce chapitre nous nous concentrons sur la recherche de patches. Nous présentons une recherche qui est une bonne approximation de la recherche exhaustive globale et nous montrons son impact sur le débruitage vidéo. Cela permet pour la première fois d'estimer l'impacte de la recherche globale sur les performances de débruitage vidéo.

La Figure 14 montre que la méthode proposée a un comportement similaire à une recherche exhaustive globale pour une fraction du coût de calcul.

3. Une méthode de débruitage vidéo récursive: NL-Kalman

Contrairement aux méthodes présentées dans les Chapitres 1 and 2, nous proposons ici une méthode récursive et causale dans ce chapitre. Cela signifie que la méthode n'utilise que l'image vidéo

Figure 15: Résultats de débruitage de séquences corrompues par un bruit gaussien avec un écart-type de 30. De gauche à droite: Débruitage avec NL-Kalman, NL-Kalman avec un flot optique oracle, VBM3D, VBM4D et l'image de référence. De haut en bas: Extrait de la séquence `crowd_run` et `pedestrian_area`. La méthode permet de récupérer plus de détails comme par exemple dans l'arbre ou dans le texte.

actuelle et ainsi que la version débruitée de l'image vidéo précédente. Le but de ces méthodes est de réduire l'écart entre les méthodes de type filtrage temporel qui sont légères et les méthodes non-causales basées sur l'auto-similarité qui, bien que très efficaces, sont beaucoup plus coûteuses.

La méthode proposée, appelée NL-Kalman, considère que la vidéo est un ensemble de trajectoire de patches qui peuvent se superposer. Selon une approche bayesienne, chaque trajectoire est modélisée comme un modèle linéaire dynamique gaussien et débruitée par un filtre de kalman. Pour estimer les différents paramètres, les patches similaires sont regroupés et on considère que leurs trajectoires partagent les mêmes paramètres. Le filtrage est principalement temporel. L'information spatiale similaire n'est utilisée que pour estimer les paramètres. Cette méthode permet d'obtenir des résultats comparables (en PSNR et SSIM) aux méthode de l'état de l'art utilisant plusieurs images vidéo pour chaque image débruitée. La méthode proposée a cependant une meilleure consistance temporelle. Bien que la consistance temporelle ne peut pas être montrée sans la vidéo, la Figure 15 montre que cette méthode préserve plus de détails que les autres.

4. Débruitage vidéo rapide sur carte graphique

Nous montrons dans les chapitres précédents que le débruitage est une étape essentielle de n'importe quelle chaîne de traitement de l'image ou de la vidéo. Malheureusement, en général à cause de contraintes de calculs, les méthodes modernes de débruitage sont rarement considérées pour être intégrée dans les chaînes de traitements. Ces algorithmes n'ont souvent que des implémentations processeur ou des implémentations carte graphique sous-optimales. Dans ce chapitre nous proposons une nouvelle implémentation carte graphique efficace de NL-means and BM3D et pour la première fois, à notre connaissance, l'algorithme de débruitage vidéo VBM3D. L'efficacité de ces implémentations permet de les utiliser dans des scénarios temps-réel. La Table 1 montre l'impact d'une implémentation optimisée pour le débruitage vidéo.

Méthode	IS	GPU	PSNR	Temps/image
NLM ours	yes	yes	31.54	0.851 ms
BM3D ours	yes	yes	32.93	4.51 ms
NLM video ours	no	yes	33.53	10.6 ms
VBM3D [EA20]	no	no	34.21	2.90 s
VBM3D ours	no	yes	34.31	3.30 ms

Table 2: Comparaison de la performance de débruitage (PSNR moyen) et du temps de calcul moyen par image pour différentes implémentations de BM3D, VBM3D et NL-means sur la base Derf (écart-type du bruit de 20). Il est indiqué si la méthode n'utilise qu'une seule image (IS) ou sur carte graphique (GPU).

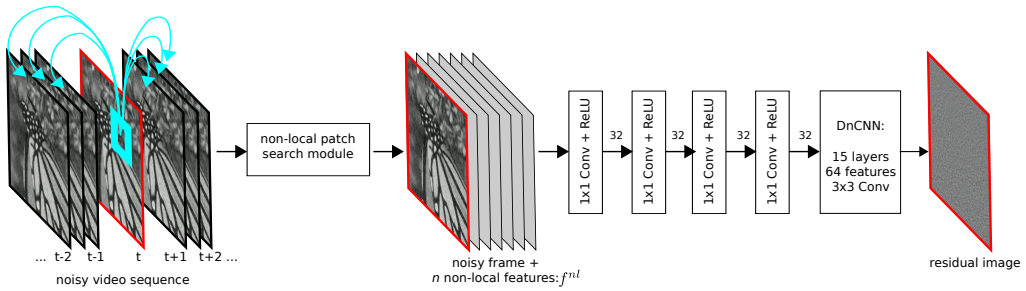


Figure 16: L'architecture du réseau de neurones proposé. Le premier module effectue une recherche de patches similaires à travers différentes images vidéo. L'image vidéo de référence ainsi que l'information de similarité (donnée sous la forme d'un vecteur des pixels centraux des patches trouvés) sont données au réseau. Les quatre premières couches du réseau sont des séries de 32 convolutions de taille 1×1. Le résultat de ces convolutions est ensuite l'entrée d'un réseau de type DnCNN [ZZC+17a] simplifié avec 15 couches.

5. Débruitage vidéo avec un réseau de neurones convolutionnel basé sur l'auto-similarité

Les méthodes à patches étaient jusqu'à récemment l'état de l'art pour le débruitage d'image mais ont depuis été dépassées par l'apprentissage et en particulier les réseaux convolutionnels. Ces méthodes sont cependant encore l'état de l'art pour le débruitage vidéo car la redondance de l'information dans la vidéo est très important pour atteindre de bonnes performances de débruitages. Le problème est que les architectures sont en général incompatibles avec l'ajout d'information auto-similaire.

Dans ce chapitre nous proposons une nouvelle manière d'intégrer de l'information auto-similaire dans un réseau de neurone convolutionnel. L'information non-locale est introduite dans le réseau grâce à une première couche, qui n'est pas apprise, qui cherche les patches les plus proches dans une région d'intérêt pour chaque patche de l'image de référence. Les pixels centraux de ces patches sont ensuite fusionnés en un vecteur associé à chaque pixel de l'image. Cette information est ensuite donnée à un réseau convolutionnel qui est entraîné pour prédire l'image débruité. Nous avons appliqué l'approche proposée aux images et aux vidéos. Pour les vidéos, les patches sont cherchés dans un volume spatio-temporel. L'architecture proposée, montrée dans la Figure 16, atteint les performances de l'état de l'art.

Figure 17: À partir du *même* point de départ et en utilisant *uniquement* la vidéo disponible, notre méthode de raffinage de réseau de neurones est capable d'apprendre à débruiter différents types de bruit sans produire d'artefacts. Les images du haut sont les images bruitées et celles du bas les débruitées. De gauche à droite: bruit gaussien, poisson, poivre et sel et bruit gaussien compressé par JPEG.

6. Débruitage vidéo sans connaître le modèle du bruit grâce un apprentissage image vidéo à image vidéo

Dans les chapitres précédents, le modèle de bruit a toujours été considéré connu et supposé qu'il n'évoluait pas dans le temps. Cependant, modéliser la chaîne de traitement qui a produit une vidéo est une tâche de rétro-ingénierie difficile, même lorsque la caméra est connue. Cela rend le traitement de la vidéo une tâche encore plus difficile. Nous proposons dans ce chapitre une méthode de débruitage qui ne dépend pas du modèle de bruit, avec une version en ligne et une version hors ligne. Cela est possible par le raffinage d'un réseau pré-entraîné pour du bruit gaussien blanc sur une vidéo grâce une stratégie d'apprentissage image vidéo à image vidéo. Notre débruiteur peut être utilisé sans avoir aucune connaissance sur la provenance de la vidéo et des traitements appliqués après l'acquisition. Le processus en ligne n'a besoin que de quelques images vidéo avant d'arriver à produire des résultats agréables visuellement pour un large éventail de dégradations. Cette méthode atteint même les performances de l'état de l'art pour du bruit gaussien, et peut être utilisée hors ligne pour encore de meilleurs résultats. La flexibilité de la méthode proposée est illustrée dans la Figure 17.

7. Démosaïquage par méthode à apprentissage profond

La plupart des caméras ne capturent qu'une seule couleur par pixel. L'image ainsi produite est alors mosaïquée et doit être interpolée pour avoir les trois couleurs pour chaque pixel. Cette étape de passage d'une image mosaïquée à une image couleur est appelée démosaïquage. Ce chapitre étudie deux récentes méthodes de démosaïquage basées sur des réseaux de neurones convolutionnels qui produisent des résultats sans aucun artefacts: Deep joint demosaicking and denoising de Gharbi *et al.* [GCPD16] et Color image demosaicking via deep residual learning de Tan *et al.* [TZZZ17]. Ces méthodes sont meilleures de presque deux décibels que les méthodes qui ne sont pas basées sur les réseaux de neurones, tout en étant plus rapide d'un ordre de magnitude. Un exemple très difficile de démosaïquage est montré dans la Figure 18. Pour cet exemple, seules les méthodes basées sur les réseaux de neurones produisent de bons résultats. Cela sonne le glas pour les méthodes qui ne sont pas basées sur l'apprentissage pour le démosaïquage.

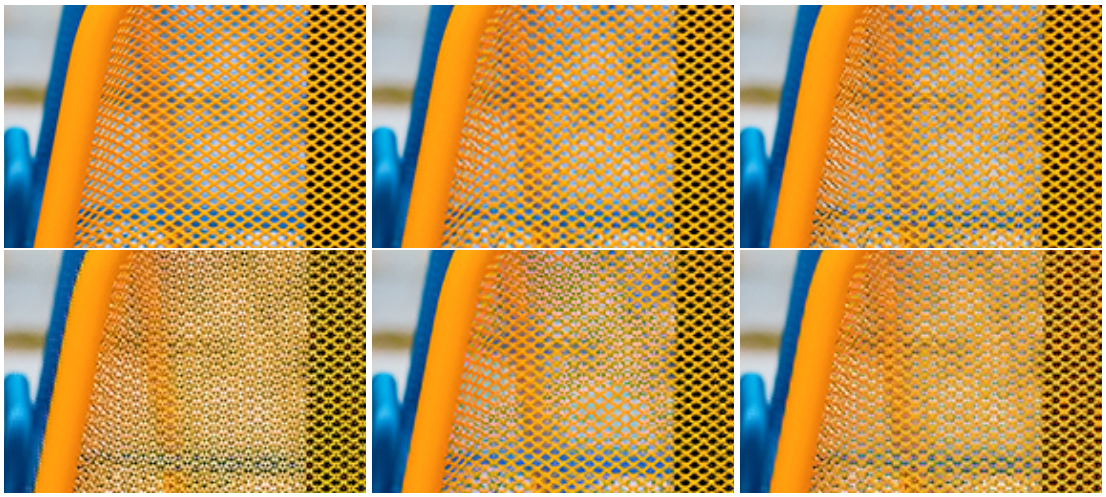


Figure 18: De haut en bas et de gauche à droite: Référence, Gharbi *et al.* [GCPD16], Tan *et al.* [TZZZ17], MLRI [KMTO14], ARI [MKTO15], Getreuer [Get12]. Les méthodes basées sur l'apprentissage produisent moins d'artefacts tout en ayant de bons temps de calculs.

Figure 19: En utilisant une acquisition rafale, notre méthode de raffinage de réseau de neurones (en utilisant comme point de départ le réseau de Gharbi *et al.* [GCPD16]) est capable non seulement de débruiter correctement ($\sigma = 5$) mais aussi de ne pas produire d'artefacts visuels comme le *zipper* ou *moiré* dans les régions difficiles.

8. Démosaïquage et débruitage conjoint grâce à un raffinage d'entraînement sur des séquences rafales brutes

Le démosaïquage, étudié dans le Chapitre 7, et le débruitage sont les deux premières étapes de la chaîne de traitement d'image de n'importe quelle caméra. Elles sont la clé pour avoir des images couleurs de bonne qualité. Un nouvel axe prometteur de recherche propose de résoudre les deux problèmes à la fois en utilisant des réseaux de neurones convolutionnels. Étant donné qu'il est très difficile, voire impossible, d'obtenir une vérité terrain pour des images naturels, ces réseaux ne sont pas entraînes avec des données brutes réelles. À la place, des données simulées sont utilisées.

Dans ce chapitre nous présentons une méthode qui permet d'apprendre un réseau directement à partir d'images brutes encore mosaïquées, sans avoir besoin de vérité terrain couleur. Nous appliquons cette méthode pour apprendre un démosaïquage et débruitage conjoint directement depuis des données brutes. Nous montrons aussi que cela peut être appliqué pour raffiner un réseau sur une acquisition rafale spécifique pour améliorer la qualité globale de la restauration. Un exemple sur des données synthétiques est présenté dans la Figure 19.

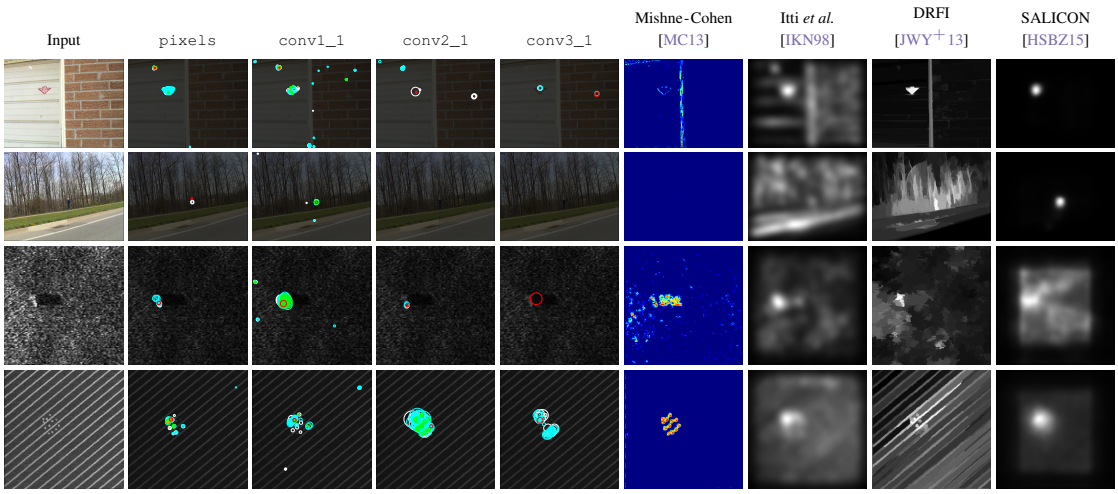


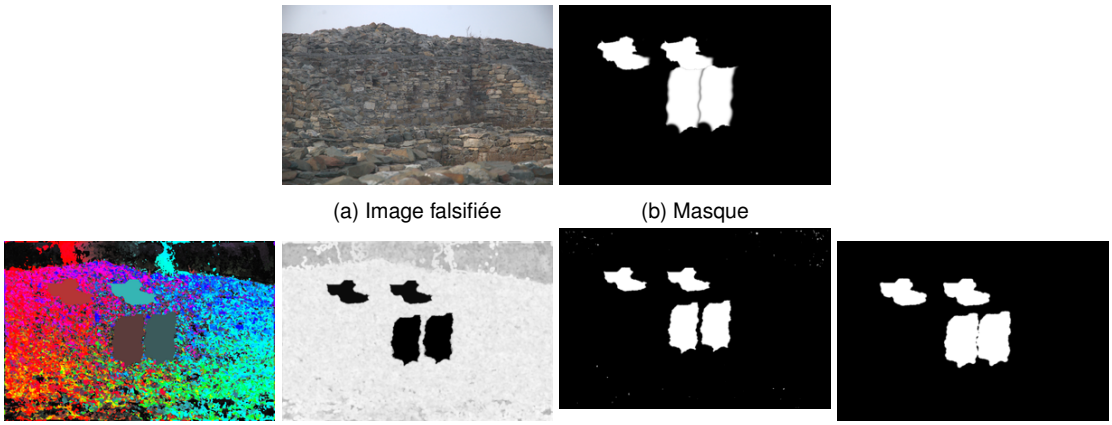
Figure 20: Exemples de détections pour notre méthode en utilisant soit directement les pixels de l'image, soit en utilisant les cartes d'activations du réseau de neurones VGG pour différentes couches (conv1_1, conv2_-1, conv3_1) avec une comparaison à [MC13], [IKN98], [JWY+13] et [HSBZ15] sur des exemples réels. Les deux premières images (première et deuxième lignes) font parties de la base de donnée Toronto [BT06], les exemples de la troisième et quatrième lignes proviennent de [MC13] et [TH99] respectivement.

9. Comment ramener le problème de détection d'anomalies dans les images à un problème de détection d'anomalies dans du bruit

Bien que le débruitage soit important pour améliorer la qualité des images et des vidéos, il peut aussi avoir d'autres applications surprenantes pour des problèmes qui, à première vue, semblent n'avoir aucun lien avec le débruitage. Une de ces applications surprenantes est la détection d'anomalies. La détection d'anomalies essaye de résoudre le problème difficile de détecter automatiquement toutes régions sortant de l'ordinaire dans une image de fond qui peut être de sources variées comme du textile ou une mammographie. Des milliers de méthodes de détection existent parce que chaque problème nécessite une modélisation différente du fond. Après avoir analysé les approches existantes, nous montrons dans ce chapitre que ce problème peut être ramené à la détection dans des images résiduelles (extraites de l'image cible) dans laquelle du bruit et les anomalies dominent. De cette manière, le problème général et difficile de la modélisation du fond est remplacé par un modèle de bruit beaucoup plus simple et qui permet de calculer des seuils de détection fiables. De ce fait, notre méthode est complètement non-supervisée et peut être appliquée à n'importe quelles images. L'image résiduelle peut aussi être calculée en utilisant des réseaux de neurones. Notre détecteur fonctionne grâce à la théorie de détection *a contrario* qui permet d'éviter la sur-détection en prenant en compte le nombre de tests dans la définition des seuils de détection. La Figure 20 montre les résultats de différentes méthodes de détection d'anomalie sur des images naturelles.

10. Analyse de PatchMatch

Comme vu dans les chapitres précédents, de nombreux problèmes du traitement d'image et de la vidéo et de la vision par ordinateur nécessitent le calcul de tous les k-plus proches voisins (k-NNF) entre deux images. Pour chaque patche de l'image requête, le k-NNF donne la position des k patches les plus similaires dans l'image référence. Grâce à l'algorithme *PatchMatch* [BSFG09a], il est possible d'approximer efficacement la recherche des plus proches voisins avec une technique de recherche collaborative qui exploite la cohérence locale des patches de



(a) Image falsifiée (b) Masque

(c) Carte de déplacement (d) Carte d'erreurs (e) Masque initial de détection (f) Masque final de détection

Figure 21: Détection de falsification en utilisant PatchMatch et des moments de Zernike pour une falsification de type copié-collé avec une translation.

l'image. Après sa présentation, de nombreuses variantes de *PatchMatch* ont été proposées. Certaines permettent même de réduire le temps de calcul par deux ordres de grandeur. Dans ce chapitre, nous étudions la convergence de *PatchMatch* et de ses variantes et proposons des bornes sur leurs vitesses de convergence. Pour cela, nous considérons une version générique de *PatchMatch* à partir de laquelle il est possible d'exprimer la majorité des variantes trouvées dans la littérature. Nous calculons aussi de manière explicite les bornes pour deux cas particuliers: la version originale de *PatchMatch* et Coherence Sensitive Hashing [KA11]. Les bornes proposées sont ensuite validées en les comparant à la vitesse de convergence observée en pratique.

11. Détection de falsifications de type copié-collé basée sur PatchMatch

Ce chapitre présente une application de la méthode PatchMach étudiée en détail dans le Chapitre 10. La méthode étudiée dans ce chapitre a été présentée dans [CPV15]. Elle détecte les falsifications en utilisant un ensemble dense de descripteurs choisis pour être invariant par rotation, par exemple les moments de zernike. Une mise en correspondance efficace des descripteurs est ensuite faite grâce à PatchMatch. Ceci est très efficace pour détecter les régions copiées. Les régions trouvées par PatchMatch sont ensuite traitées pour trouver les détections finales. Cela permet une détection précise des régions falsifiées à l'intérieur d'une seule image suspicieuse. Nous étendons aussi la méthode pour pouvoir utiliser des descripteurs SIFT dense ce qui permet aussi de détecter les falsifications lors de modifications de type "Poisson editing". Un exemple de détection est montré dans la Figure 21.

12. Détection de falsifications de type copié-collé basée sur des descripteurs parcimonieux

La méthode présentée dans le Chapitre 11 surdétecte les régions similaires sans qu'elles soient falsifiées. Cela est due à la difficulté du problème et en particulier parce que les images peuvent contenir des objets similaires sans forcément avoir de falsifications. Ces objets similaires sont détectés parce que la plupart des méthodes ne sont pas assez robustes pour pouvoir discerner entre des copies et des objets similaires. La question qui se pose est comment ne pas détecter

Figure 22: À gauche, une image avec deux objets similaires mais qui ne sont pas des copies. À droite, une image où les objets sont des copies. Les patches montrées en dessous de chaque image correspondent aux carrés rouges. Ces patches montrent qu'une différence est visible à cette échelle et que les descripteurs peuvent être différenciés. C'est pour cela que la méthode peut discerner entre des copies ou juste des objets similaires.

les régions qui sont naturellement similaires tout en détectant les régions falsifiées. D'autant plus que les copies peuvent avoir été faites après rotation, changement d'échelle et suivie parfois par un ajout de bruit ou une compression comme JPEG par exemple.

C'est pour cette raison que nous proposons une nouvelle méthode basée une comparaison de patches similaire à celles de la Partie I. Nous utilisons des poins-clés de type SIFT, c'est-à-dire parcimonieux et avec une invariance à la rotation, au changement d'échelle et aux changements d'illumination. Pour discerner entre falsification et similarité naturelle, nous introduisons une comparaison de patches *a contrario* qui permet d'avoir des garanties théoriques sur le nombre de fausses alarmes. Cette méthode est ensuite validée sur plusieurs bases d'images. Étant donné que la méthode est complètement non-supervisée, elle peut être intégrée dans une chaîne de détection automatique d'images falsifiées. Avec la Figure 22, nous montrons qu'il est possible de différencier des objets naturellement similaires de falsifications en n'utilisant que des patches locaux.

13. Publications

Ce travail a mené aux publications suivantes:

- **Global patch search boosts video denoising**, Thibaud Ehret, Pablo Arias and Jean-Michel Morel, *International Conference on Computer Vision Theory and Applications, 2017*

- **Automatic Detection of Internal Copy-Move Forgeries in Images**, Thibaud Ehret, *Image Processing On Line, 2018*
- **Model-blind video denoising via frame-to-frame training**, Thibaud Ehret, Axel Davy, Jean-Michel Morel, Gabriele Facciolo and Pablo Arias, *Conference on Computer Vision and Pattern Recognition, IEEE, 2018*
- **Non-Local Kalman: A recursive video denoising algorithm**, Thibaud Ehret, Jean-Michel Morel and Pablo Arias, *International Conference on Image Processing, IEEE, 2018*
- **On the convergence of PatchMatch and its variants**, Thibaud Ehret and Pablo Arias, *Conference on Computer Vision and Pattern Recognition, IEEE, 2018*
- **Reducing Anomaly Detection in Images to Detection in Noise**, Axel Davy, Thibaud Ehret, Jean-Michel Morel and Mauricio Delbracio, *International Conference on Image Processing, IEEE, 2018*
- **A Non-Local CNN For Video Denoising**, Axel Davy, Thibaud Ehret, Jean-Michel Morel, Pablo Arias and Gabriele Facciolo, *International Conference on Image Processing, IEEE, 2019*
- **Video Denoising by Combining Patch Search and CNNs.**, Axel Davy, Thibaud Ehret, Jean-Michel Morel, Pablo Arias and Gabriele Facciolo, *in Journal of Mathematical Imaging and Vision, Springer, 2021*
- **A Study of Two CNN Demosaicking Algorithms**, Thibaud Ehret and Gabriele Facciolo, *Image Processing On Line, 2019*
- **How to Reduce Anomaly Detection in Images to Anomaly Detection in Noise**, Thibaud Ehret, Axel Davy, Jean-Michel Morel and Mauricio Delbracio, *Image Processing On Line, 2019*
- **Image Anomalies: A Review and Synthesis of Detection Methods**, Thibaud Ehret, Axel Davy, Jean-Michel Morel and Mauricio Delbracio, *in Journal of Mathematical Imaging and Vision, Springer, 2019*
- **Joint Demosaicking and Denoising by Fine-Tuning of Bursts of Raw Images**, Thibaud Ehret, Axel Davy, Pablo Arias and Gabriele Facciolo, *International Conference on Computer Vision, IEEE, 2019*
- **GPU acceleration of NL-means, BM3D and VBM3D**, Axel Davy and Thibaud Ehret, *Journal of Real-Time Image Processing, Springer, 2020*

Part I

Patch-based video denoising

1 VBM3D and its extensions

VBM3D is a popular video denoising method. It is an extension to video of the well-known image denoising algorithm BM3D, which takes advantage of the sparse representation of stacks of similar patches in a transform domain. The extension is rather straightforward: the similar 2D patches are taken from a spatio-temporal neighborhood which includes neighboring frames. In spite of its simplicity, the algorithm offers a good trade-off between denoising performance and computational complexity. In this chapter we revisit this method, providing an open-source C++ implementation reproducing the results. A detailed description is given and the choice of parameters is thoroughly discussed. Furthermore, we study and compare several extensions of the original algorithm: (1) a multi-scale implementation, (2) the use of 3D patches, (3) the use of optical flow to guide the patch search. These extensions allow to obtain results which are competitive with even the most recent state of the art.

1.1 Introduction

VBM3D was proposed by [DFE07] as an adaptation to video denoising of BM3D, the successful image denoising algorithm [DFKE07b]. The algorithm is designed for additive white Gaussian noise (AWGN) with zero mean and standard deviation σ, *i.e.*

$$v(x) = u(x) + n(x), \quad n(x) \sim \mathcal{N}(0, \sigma^2),$$

where x is a spatio-temporal position in the video domain (a pixel), v is the noisy video and u the unknown clean video. We consider that the v consists of $f + 1$ frames written v_i for i between 0 and f. A patch is a small rectangular piece of the video, for example of size $8 \times 8 \times 3$ (8 pixels width and height, 3 pixels in the temporal dimension). Patches are represented by bold lowercase letters, *e.g.* $\boldsymbol{p}$. If we need to emphasize the location of the patch we write $\boldsymbol{p}(x)$, where x represents the location on the video of the top-left-front pixel of the patch.

The denoising principle of VBM3D (and BM3D) is based on the redundancy of similar patches. Groups of similar 2D patches are assembled in a 3D stack of patches. A separable 3D transform is applied to this stack. The stack is denoised by applying a shrinkage operator to the coefficients in the transformed domain. The algorithm follows four basic steps:

1. Search for similar patches in the sequence, grouping them in a 3D stacks,
2. Apply a 3D linear domain transform to the 3D block,
3. Shrink the transformed coefficients,
4. Apply the inverse transform,

5. Aggregate the resulting patches in the video.

The underlying idea here is that, due to the high redundancy of a stack of similar patches, the energy will be concentrated in a few coefficients of the transform, while the noise is spread evenly among all coefficients. This allows to jointly denoise the patches of each stack. The reconstruction of the estimated video is obtained by aggregating for each pixel the estimated patches that contain it. This principle is applied twice. The first time the patches are denoised using a hard threshold in the transformed domain. In the second iteration a Wiener filter is used, with Wiener coefficients computed using the output of the first step as oracle. For the parameters of the method we will use the same notation as in [DFE07]. The different parameters for the patch search are the number N of patches to return, the number of temporal frames N_f that are being searched, the search region for the reference frame N_s, the search region for the other frames N_{pr}, the maximum number N_b of patches to compute for each local search, a correcting factor d and a maximum distance threshold τ. The threshold parameter in the first step is λ_{3D}. We also write $\otimes$ for the element-wise product. Similar parameters will be used for the hard thresholding first step and for the Wiener filtering second step. Because they can have different values for the different steps, they'll be differentiated using the subscript $hard$ and $wien$ respectively.

There exist other adaptations to video and 3D images of this framework: BM4D [MKEF13] and VBM4D [MBFE12]. These methods stack similar 3D spatio-temporal patches in a 4D stack. The main difference between them is that VBM4D uses motion-compensated patches whereas BM4D aims at denoising volumetric images (such as those appearing in medical imaging), where the 3rd dimension is just another spatial dimension. In [AFM18a] however, it was shown that even non-motion-compensated 3D patches provide a very good denoising performance.

In this chapter, we revisit the VBM3D method, providing an open source implementation and we discuss some variants introduced in [AFM18a], such as 3D patches (as in BM4D [MKEF13]), optical flow guided patch search, and a multiscale implementation. In the next section, the algorithm itself is reviewed. In the following sections we consider three possible extensions: multiscale, spatio-temporal patches and motion-compensated patch search using an optical flow. Finally the performance of the algorithm is compared against other state of the art algorithms.

1.2 VBM3D : the algorithm

VBM3D performs two steps, as its image counterpart BM3D. The first *hard-thresholding* step computes a a basic estimate which is refined in the *Wiener filtering* step, yielding the final estimate. Figure 1.1 represents the global structure of the algorithm. The pseudocode of the core of the algorithm is presented in Algorithm 5.

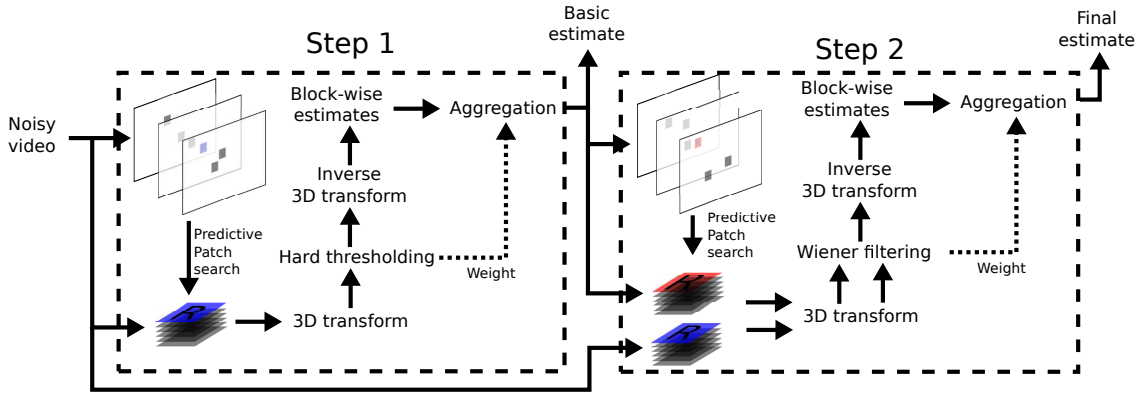


Figure 1.1: Scheme of the core of the VBM3D algorithm.

1.2.1 The patch search

The groups of similar patches are built by selecting a reference patch and searching around it for its nearest neighbors. This patch search is the main difference between BM3D and VBM3D. The image version of the algorithm searches a square 2D window centered at the reference patch. While the same could be done in a space-time volume for videos, Dabov et al. [DFE07] propose a *predictive search* heuristic to reduce the size of the search window. The idea behind the proposed search is to track patches in the video. Let $\boldsymbol{p}$ and $\boldsymbol{q}$ be two patches at positions (x, y, t) and (x', y', t') respectively. The distance between $\boldsymbol{p}$ and $\boldsymbol{q}$ used to find the nearest neighbors for a given regularizing factor δ is

$$d(\boldsymbol{p}, \boldsymbol{q}) = \begin{cases} \|\boldsymbol{p} - \boldsymbol{q}\|_2^2 - \delta \text{ if } x = x' \text{ and } y = y' \\ \|\boldsymbol{p} - \boldsymbol{q}\|_2^2 \text{ otherwise} \end{cases} . \tag{1.1}$$

This distance regularizes the patch trajectories by assuming that the patch does not move. Suppose that the reference patch is $\boldsymbol{p}$ located at (x, y, t). The goal of the proposed search is to find (at most) N patches in a window of $2N_f + 1$ frames around frame t. The steps are the following:

1. Find the N_b nearest neighbors to $\boldsymbol{p}$ in frame t in a square search region of size $N_s \times N_s$ centered at the location of the reference patch (x, y, t). Let L_t be the set of found patches.

2. Find the N_b nearest neighbors to $\boldsymbol{p}$ in frames $t' = t + 1, ..., t + N_f$. The search region at t' is the union of $N_{pr} \times N_{pr}$ square regions centered at positions of the candidates in $L_{t'-1}$, where $N_{pr} < N_s$. This is depicted in Figure 1.2.

3. Similarly, find N_b nearest neighbors to $\boldsymbol{p}$ in frames $t' = t - 1, t - 2, ..., t - N_f$. This time the search region is the union of squares centered at the positions of the patches in $L_{t'+1}$.

4. Remove the candidates with distance (1.1) larger than τ from

$$L = \bigcup_{t'=t-N_f}^{t+N_f} L_{t'},$$

   the set combining all the candidates computed in each frame.

5. Keep the best N candidates from L.

6. Because the next steps of the algorithm (in particular the domain transforms) require a number of patches that is a power of 2, only the largest power of 2 smaller or equal to N is kept.

These steps are summarized in Algorithms 1 and 2.

1.2.2 Patch stack filtering: hard-threshold step

The first step processes each frame by filtering stacks of similar patches and aggregating them in an output image. The reference patches for the stacks are all patches on a sub-grid of step st_{hard}. For each group there is first an estimation, followed by an aggregation.

Algorithm 1: `compute_similar_patches`: VBM3D patch search

input : Noisy video $v = (v_0, \dots, v_f)$, a reference patch $\boldsymbol{p}$, the number of patches to return N, a number of temporal frames N_f, a search region for the reference frame N_s, a search region for the other frames N_{pr}, the maximum number of patches to compute with each local search N_b, the size of the patch k, a correcting factor d, a maximum distance threshold τ

output : L the list of the N patches closest to $\boldsymbol{p}$ and their distance

$t \leftarrow$ frame at which $\boldsymbol{p}$ is located

$L_t \leftarrow$ `local_search`$(\boldsymbol{p}, \boldsymbol{p}, N_s, k, N_b, d, u_t)$ `// Centered on` $\boldsymbol{p}$ `using` $\boldsymbol{p}$ `as a reference`

`// Search in the` N_f `following frames`

for $t_f = (t+1)$ *to* $\min(t + N_f, f - 1)$ **do**

  for $\boldsymbol{q} \in L_{t_f - 1}$ **do**

    $L_{t_f} \leftarrow$ `local_search`$(\boldsymbol{q}, \boldsymbol{p}, N_{pr}, k, N_b, d, u_{t_f})$ `// Centered on` $\boldsymbol{q}$ `using` $\boldsymbol{p}$ `as a reference`

`// Search in the` N_f `previous frames`

for $t_p = t - 1$ *to* $\max(t - N_f, 0)$ **do**

  for $\boldsymbol{q} \in L_{t_p + 1}$ **do**

    $L_{t_p} \leftarrow$ `local_search`$(\boldsymbol{q}, \boldsymbol{p}, N_{pr}, k, N_b, d, u_{t_p})$ `// Centered on` $\boldsymbol{q}$ `using` $\boldsymbol{p}$ `as a reference`

$L \leftarrow \bigcup_{i \in [\![t - N_f; t + N_f]\!]} L_i$

$L \leftarrow$ elements from L with distance smaller than τ

`// the 3D transform requires a number of patches which is a power of 2`

$N_l \leftarrow$closest power of 2 small or equal than $min(sizeofL, N)$

if *L has more than N_l elements* **then**

  $L \leftarrow N_l$ best candidate from L

return L

Algorithm 2: `local_search`: Local search

input : A patch $\boldsymbol{p}$ center of the search region, a reference patch $\boldsymbol{p}'$, the search region size s, the size of the patch k, the number of patches to return N_b, a correcting factor d, a frame u

output : L the list of the patches closest to $\boldsymbol{p}'$ in the region described by $\boldsymbol{p}$ and s and their distance

`// Compute the distances for all the patches in the local region`

for *each patch $\boldsymbol{q}$ of size $k \times k$ in the spatial region of size $s \times s$ centered on $\boldsymbol{p}$* **do**

  if *$\boldsymbol{q}$ is at the same spatial position than $\boldsymbol{p}$* **then**

    Add $(\|\boldsymbol{q} - \boldsymbol{p}'\|_2^2 - d, \boldsymbol{q})$ to L

  else

    Add $(\|\boldsymbol{q} - \boldsymbol{p}'\|_2^2, \boldsymbol{q})$ to L

Sort L according to the value of the distance

Keep the N_b best elements in L

return L

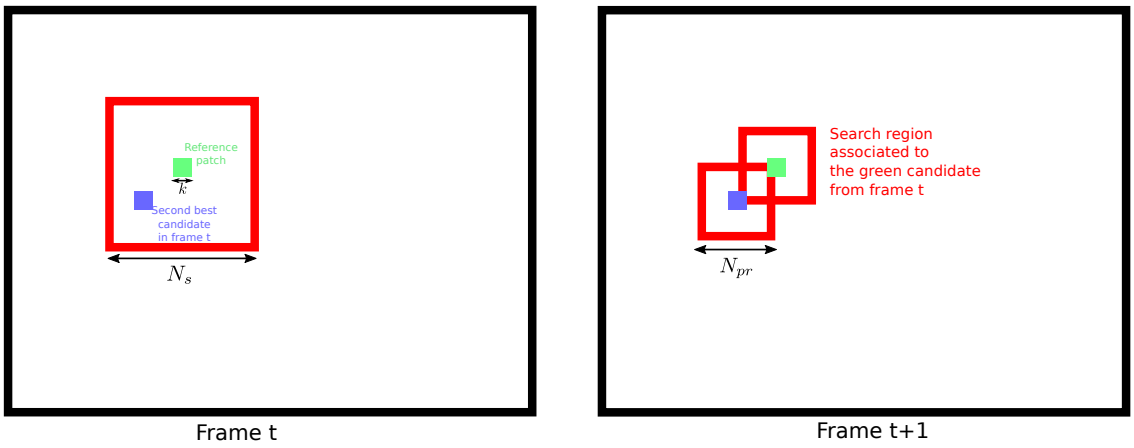


Figure 1.2: The patch search starts with a local spatial search in the current frame of size $N_s \times N_s$. Only the N_b best candidates are then used in the following frame to predict where to search. This leads to N_b local spatial searches (each centered at the candidates from the previous frame), of size $N_{pr} \times N_{pr}$. In practice, $N_s = 7$, $N_{pr} = 5$ and $N_b = 2$.

Estimation. For a given reference patch $\boldsymbol{p}$, we first find its similar patches L as described in Section 1.2.1. These patches are stacked in a 3D volume $\mathcal{P}(\boldsymbol{p})$, of size $k \times k \times N$. The first spatial slice of this stack is the reference patch, and the remaining ones are the similar patches in L ordered by their distance to $\boldsymbol{p}$. A 3D domain transform is first applied to the group of patches followed by a thresholding of the resulting spectrum (excepting the DC component of every patch). The inverse 3D domain transform is then applied to the thresholded coefficients to obtain the estimation, *i.e.*

$$\widehat{\mathcal{P}(\boldsymbol{p})} = T_{3D}^{-1}(HT_{\lambda}(T_{3D}(\mathcal{P}(\boldsymbol{p})))) \tag{1.2}$$

where HT is defined by

$$HT_{\lambda\sigma}(\nu) = \begin{cases} \nu \text{ if } \nu \text{ is a DC component or } |\nu| > \lambda\sigma \\ 0 \text{ otherwise.} \end{cases} \tag{1.3}$$

In practice the 3D transforms are chosen to be separable, consisting of a 2D spatial transform applied directly on the patches of $\mathcal{P}(\boldsymbol{p})$, typically a bi-orthogonal wavelet transform, followed by an 1D transform along the third dimension of the stack, typically a Haar transform. The pseudocode of the estimation for the first step is presented in Algorithm 3.

Aggregation. An output pixel is estimated several times, since it belongs to several patches and each patch can be estimated multiple times (once for each group it belongs to). To compute the output frame $\widehat{u}$ these estimates are aggregated. For a pixel ρ of the frame, this aggregation is performed as

$$\widehat{u}(\rho) = \frac{\sum_{\text{patch } \boldsymbol{p} \text{ of } v} \sum_{\boldsymbol{q} \in \widehat{\mathcal{P}(\boldsymbol{p})}} \omega_{\boldsymbol{p}}(\rho)\boldsymbol{q}(\rho)}{\sum_{\text{patch } \boldsymbol{p} \text{ of } v} \omega_{\boldsymbol{p}}(\rho)} \tag{1.4}$$

where a patch $\boldsymbol{p}$ has a support of the size of the image and is zero everywhere except on the actual location of the patch. The weights ω used during the aggregation have two contributions. A "per-stack" contribution w_{ht} computed during the estimation is $w_{ht}(\boldsymbol{p}) = \frac{1}{\sigma^2 N_{hard}(\boldsymbol{p})}$ where $N_{hard}(\boldsymbol{p})$ corresponds to the number of coefficients that have not been thresholded during the estimation of $\widehat{\mathcal{P}(\boldsymbol{p})}$. Each coefficient $w_{ht}(\boldsymbol{p})$ is properly defined because $N_{hard}(\boldsymbol{p})$ is always positive as the DC component is never thresholded. These weights penalize estimates with large variance, as proposed in [Gul03]. In addition a "per-pixel" aggregation weight is used to smooth out the

blocking artifacts that result from working with patches. This weight comes from a 2D Kaiser window $K(\boldsymbol{p})$ of size $k_{hard} \times k_{hard}$ (see Eq. (1.6)) located on the position of $\boldsymbol{p}$. The Kaiser window of size $L_x \times L_y$ of parameter β is

$$K(\boldsymbol{p})(x,y) = I_0\left(\beta\sqrt{1-(2x/L_x)^2}\right) I_0\left(\beta\sqrt{1-(2y/L_y)^2}\right) / I_0(\beta)^2 \tag{1.5}$$

$$\text{with } 0 \leqslant x \leqslant L_x, 0 \leqslant y \leqslant L_y \text{ and } I_0 \text{ the zeroth-order modified Bessel function} \tag{1.6}$$

Finally the (patch) weight is defined by $\omega_{\boldsymbol{p}} = w_{ht}(\boldsymbol{p})K(\boldsymbol{p})$. The pseudocode with the aggregation step can be found in Algorithm 5.

Algorithm 3: `ht_filtering`: Hard thresholding

input : A group of similar patches L, a 3D transform T_{3D}, the noise variance σ^2
output : A list of filtered patches $\hat{L}$, the aggregation weight ω

$L \leftarrow T_{3D}(L)$
$n \leftarrow 0$
for *each patch* $\boldsymbol{p}$ *in* L **do**
  for *each pixel* ρ *of* $\boldsymbol{p}$ **do**
    if $\boldsymbol{p}(\rho) > \lambda\sigma$ *or* ρ *is the DC component* **then**
      $n \leftarrow n + 1$
      $\hat{\boldsymbol{p}}(\rho) \leftarrow \boldsymbol{p}(\rho)$
    else
      $\hat{\boldsymbol{p}}(\rho) \leftarrow 0$
$\omega = \frac{1}{\sigma^2 n}$
$\hat{L} \leftarrow T_{3D}^{-1}(L)$
return $\hat{L}, \omega$

Algorithm 4: `wiener_filtering`: Wiener thresholding

input : A group of similar patches L, a first estimate of L called L', a 3D transform T_{3D}, the noise variance σ^2
output : A list of filtered patches $\hat{L}$, the aggregation weight ω

$L \leftarrow T_{3D}(L)$
$\omega \leftarrow 0$
for *each patch* $\boldsymbol{p}$ *in* L, $\boldsymbol{p}'$ *the corresponding patch in* L' **do**
  for *each pixel* ρ *of* $\boldsymbol{p}$ **do**
    $\alpha \leftarrow \frac{\boldsymbol{p}'(\rho)^2}{\boldsymbol{p}'(\rho)^2+\sigma^2}$
    $\hat{\boldsymbol{p}}(\rho) \leftarrow \alpha\boldsymbol{p}(\rho)$
    $\omega \leftarrow \omega + \alpha^2$
$\omega \leftarrow \frac{1}{\sigma^2\omega}$
$\hat{L} \leftarrow T_{3D}^{-1}(\hat{L})$
return $\hat{L}, \omega$

1.2.3 Patch stack filtering: Wiener filtering step

The second step of the algorithm uses the basic estimate computed during the first step for the patch search to compute the coefficients of a Wiener shrinkage of the transformed coefficients.

The video is processed by building groups of patches around reference patches from a coarse subgrid with step st_{wien}.

Estimation. Given a reference patch $\boldsymbol{p}$, a group of similar patches is selected as described in Section 1.2.1, but using patches extracted from the basic estimate for computing the patch distance. Two sets of patches are extracted: one from the noisy sequence and the other one from the basic estimate at the same locations. Each set of patches is stacked in a 3D volume: $\mathcal{P}(\boldsymbol{p})$ for the noisy patches and $\mathcal{P}(\hat{\boldsymbol{p}})$ for patches of the basic sequence. The noisy stack is denoised with a Wiener filter. The filtering coefficients are estimated from $\mathcal{P}(\hat{\boldsymbol{p}})$. Just like the first step, a 3D domain transform is first applied to the group of patches before the application of the Wiener filter. The computation is done following Equation (1.7).

$$\widehat{\mathcal{P}(\boldsymbol{p})} = T_{3D}^{-1}(WF(T_{3D}(\mathcal{P}(\boldsymbol{p})))) \tag{1.7}$$

where WF on a frequency ν (with $\hat{\nu}$ corresponding to the same frequency in the basic estimation) is defined by

$$WF(\nu) = \frac{\hat{\nu}^2}{\hat{\nu}^2 + \sigma^2} f \tag{1.8}$$

In practice the 3D transforms are chosen as a 2D transform applied directly on the patches of $\mathcal{P}(\boldsymbol{p})$ and $\mathcal{P}(\hat{\boldsymbol{p}})$, typically a DCT, followed by an 1D transform along the third dimension of the group, typically a Haar transform. The pseudocode of the estimation for the first step is presented in Algorithm 4.

Aggregation. Just as with the first step, the estimation gives multiple estimates per pixel. Therefore the different estimates are aggregated using the same principle as in Section 1.2.2, defined by Equation (1.4). The only difference lies in the weights. The weights ω used during the aggregation are computed in part during the estimation. The part w_{wf} computed during the estimation is $w_{wf}(\boldsymbol{p}) = \frac{1}{\sigma^2}\left(\sum_\rho \left(\frac{\hat{\boldsymbol{p}}_i(\rho)^2}{\hat{\boldsymbol{p}}_i(\rho)^2+\sigma^2}\right)^2\right)^{-1}$ which is actually linked to the squared ℓ_2 norm of the vector of coefficients used for the filtering. The rest of the weight come from a 2D Kaiser window $K(\boldsymbol{p})$ of size $k_{wien} \times k_{wien}$ applied to avoid boundary effects from the patches. Finally the pixel weight is defined by $\omega_{\boldsymbol{p}}(x) = w_{wf}(\boldsymbol{p})K(\boldsymbol{p})$. The pseudocode with the aggregation step can be found in Algorithm 5.

1.2.4 Reproducing the original VBM3D

We now compare the results obtained with our implementation and with the binaries released by the authors of the original VBM3D [DFE07][1]. Throughout this work we will use a test set of seven grayscale test sequences obtained from the *Derf's Test Media collection*[2]. The original sequences are of higher resolution and in RGB. They have been downscaled and converted to grayscale.

Table 1.1 compares the PSNR obtained by our implementation to that of the original binaries. Our results are slightly below: The average gap starts at 0.27dB for $\sigma = 10$ and reduces to 0.18dB for $\sigma = 40$. *Station* and *sunflower* are the sequences where the gap is larger, reaching 0.68dB for the later at $\sigma = 10$. A visual comparison of both results is shown in Figure 1.3, for noise $\sigma = 40$. Both results are visually very similar, with the same type of artifacts. A closer inspection reveals that the result of the original VBM3D is slightly smoother, resulting in less noticeable

[1] Available at http://www.cs.tut.fi/~foi/GCF-BM3D/.
[2] https://media.xiph.org/video/derf/.

Algorithm 5: VBM3D algorithm

input : A video v, the noise variance σ^2, the number of similar patches to compute N, a number of temporal frames N_f, the size of the search region in the reference frame N_s, the size of the search region in the other frame N_{pr} the number of patches kept in each frame N_b, the size of the patch k, d, the distance threshold τ, the thresholding parameter λ_{3D}, the domain transform T_{3D}, the coefficient fo the Kaiser window β, the step of the grid on which the patch are taken st

output : An final estimate denoised video $\hat{v}^{(2)}$

$K_1 \leftarrow$ Kaiser window of size k_{hard} and coefficient β_{hard}

$K_2 \leftarrow$ Kaiser window of size k_{wien} and coefficient β_{wien}

`// Step 1`

for *each* $\boldsymbol{p}$ *on the grid of step* st_{hard} **do**

`// Search for similar patches in the noisy video`

$L_{\boldsymbol{p}} \leftarrow$ `compute_similar_patches`$(v, \boldsymbol{p}, N_{hard}, N_f, N_s, N_{pr}, N_b, k_{hard}, d_{hard}, \tau_{hard})$

`// Filter the group of patches using a hard thresholding`

$(\hat{L_{\boldsymbol{p}}}, \omega) \leftarrow$ `ht_filtering`$(L_{\boldsymbol{p}}, T_{3D,hard}, \sigma^2)$

for $\boldsymbol{q} \in \hat{L_{\boldsymbol{p}}}$ **do**

$a(\boldsymbol{q}) \leftarrow a(\boldsymbol{q}) + \omega K_1 \otimes \boldsymbol{q}$

$w(\boldsymbol{q}) \leftarrow w(\boldsymbol{q}) + \omega K_1$

for *each pixel* x **do**

$\hat{v}^{(1)}(x) \leftarrow a(x)/w(x)$

`// Step 2`

for *each* $\boldsymbol{p}$ *on the grid of step* st_{wien} **do**

`// Search for similar patches in the basic estimate`

$L'_{\boldsymbol{p}} \leftarrow$ `compute_similar_patches`$(\hat{v}^{(1)}, \boldsymbol{p}, N_{wien}, N_f, N_s, N_{pr}, N_b, k_{wien}, d_{wien}, \tau_{wien})$

$L_{\boldsymbol{p}} \leftarrow$ patches from v at the same position than the one from $L'_{\boldsymbol{p}}$

`// Filter the group of patches using a Wiener filtering`

$(\hat{L_{\boldsymbol{p}}}, \omega) \leftarrow$ `wiener_filtering`$(L_{\boldsymbol{p}}, L'_{\boldsymbol{p}}, T_{3D,wien}, \sigma^2)$

for $\boldsymbol{q} \in \hat{L_{\boldsymbol{p}}}$ **do**

$a(\boldsymbol{q}) \leftarrow a(\boldsymbol{q}) + \omega K_2 \otimes \boldsymbol{q}$

$w(\boldsymbol{q}) \leftarrow w(\boldsymbol{q}) + \omega K_2$

for *each pixel* x **do**

$\hat{v}^{(2)}(x) \leftarrow a(x)/w(x)$

return $\hat{v}^{(2)}$

σ	Method	Crowd	Park	Pedestrians	Station	Sunflower	Touchdown	Tractor	Average
10	VBM3D (original)	35.65	34.75	40.83	38.93	40.49	39.04	37.01	38.10
	VBM3D (ours)	35.52	34.59	40.65	38.38	39.81	39.01	36.82	37.83
20	VBM3D (original)	32.25	31.25	36.94	35.45	36.46	36.08	33.07	34.50
	VBM3D (ours)	32.06	31.12	36.81	35.10	35.95	36.05	32.97	34.30
40	VBM3D (original)	28.65	27.68	32.81	32.02	32.65	33.52	29.41	30.96
	VBM3D (ours)	28.39	27.64	32.62	31.80	32.31	33.35	29.38	30.78

Table 1.1: Comparison of the denoising quality with the binary program provided by Dabov *et al.*with [DFKE07a]. Results were both computed using the normal profile 'np' parameters.

DCT artifacts and on some textures with decreased contrast (see for instance the grass in the rightmost figure).

Figure 1.3: Top: results of VBM3D [DFE07], with the original authors' implementation. Bottom: result obtained with our implementation. The noise level is $\sigma = 40$. The contrast has been linearly scaled for better visualization.

1.3 Extensions

1.3.1 Using optical flow to guide the search

Many video denoising methods take advantage of the optical flow to estimate motion in the video. Typically, optical flow is used by video denoising methods that require aggregating information along motion trajectories [BK95, LF10, BLM16, EMA18, AM19a]. These methods require a motion estimate as accurate as possible. Most patch-based methods, on the other hand, do not require such an accurate motion estimate, as they are based on finding similar patches in a 3D search region [BCM05b, DFE07, MBFE12]. These methods either do not use any motion estimate at all, or use a very rough one (e.g. block matching) to guide the search region. However, it has been observed that using optical flow to shape the search region can still be beneficial for patch based methods [AM18a, AFM18a], as it allows to find better matches.

For a pair of two images A and B, the optical flow aims at finding a vector field $o(x, y) = (\delta_x, \delta_y)$ such that any point (x, y) of the image domain solves

$$A(x + \delta_x, y + \delta_y) = B(x, y). \tag{1.9}$$

The displacement vector can be sub-pixel, in which case the image A needs to be interpolated. We decided to use the TVL1 optical flow method [ZPB07], in particular the implementation provided in [SPMLF13]. We compute the optical flow in a downscaled version of the video by a factor of 4, and scale it back to the original resolution. This reduces the running time and the impact of the noise while still having a reasonable precision, as shown in [EMA18].

We add the optical flow to VBM3D as a guide, the same way it is done for VNLB [AM18a]. The spatio-temporal search region is defined as two sequences of N_f square windows of size $N_{pr} \times N_{pr}$ (plus the $N_s \times N_s$ window corresponding to the reference frame), whose centers follow the motion trajectory of the reference patch. The trajectory is estimated using the forward and backward optical flow. We use the center corresponding to the position of the reference patch propagated in previous, respectively following, frames using the backward, respectively forward, optical flow. The forward half of the trajectory $\varphi_{x,y,t}$ passing through (x, y, t) is computed by integrating the forward optical flow o^f (also indexed by time on top of the spatial position) as follows:

$$\varphi_{x,y,t}(h) = o^f([\varphi_{x,y,t}(h-1)], h-1) + \varphi_{x,y,t}(h-1), \quad h = t+1, \dots, t+N_f, \tag{1.10}$$

where $[\,\cdot\,]$ and $\lfloor\cdot\rfloor$ denote the round and floor operators. The backward half of the trajectory is defined analogously using the backward optical flow o^b.

This also means that only one center needs to be tracked, in contrast to the N_b required by a regular VBM3D search. Parameters are kept the same as in the original algorithm. The guided version of the search is summarized in Algorithm 6.

We also tested setting the parameter d to zero for both steps since one can assume that it would be redundant with the regularization of the patch search offered by the optical flow. PSNR results are shown in Table 1.2. Using the optical flow as a guide greatly improves the quality of the denoising. The non-zero d yields better results both with and without the optical flow guide, thus we decided to keep it in our final version.

Figure 1.4 shows results obtained with the different extensions that will be described in this section. The first column corresponds to VBM3D with the default parameters, and the 3rd column to the one using an optical flow guided search region (denoted "VBM3D OF"). Guiding the patch search allows to recover more details. This is because the tracking of patches is more robust to motion than the block-matching suggested for the original VBM3D and therefore provides better matches. Moreover, the optical flow is not required to be very precise since it is used only as a guide for the center of the search region.

Algorithm 6: `compute_similar_patches`: VBM3D patch search guided by the optical flow

input : Noisy video $v = (v_0, \dots, v_f)$, a reference patch $\boldsymbol{p}$, the number of patches to return N, a number of temporal frames N_f, a search region for the reference frame N_s, a search region for the other frames N_{pr}, the maximum number of patches to compute with each local search N_b, the size of the patch k, a correcting factor d, a maximum distance threshold τ, the forward and backward optical flows o_f and o_b

output : L the list of the N patches closest to $\boldsymbol{p}$ and their distance

$t \leftarrow$ frame at which $\boldsymbol{p}$ is located
$L_t \leftarrow$ `local_search`$(\boldsymbol{p}, \boldsymbol{p}, N_s, k, N_b, d, u_t)$
`// Search in the` N_f `following frames`
$\boldsymbol{q} \leftarrow \boldsymbol{p}$
for $t_f = (t+1)$ *to* $\min(t + N_f, f - 1)$ **do**
  $\boldsymbol{q} \leftarrow o_f(\boldsymbol{q}, t_f - 1)$ `Follow the center using the forward optical flow`
  $L_{t_f} \leftarrow$ `local_search`$(\boldsymbol{q}, \boldsymbol{p}, N_{pr}, k, N_b, d, u_{t_f})$
`// Search in the` N_f `previous frames`
$\boldsymbol{q} \leftarrow \boldsymbol{p}$
for $t_p = t - 1$ *to* $\max(t - N_f, 0)$ **do**
  $\boldsymbol{q} \leftarrow o_b(\boldsymbol{q}, t_p + 1)$ `Follow the center using the forward optical flow`
  $L_{t_p} \leftarrow$ `local_search`$(\boldsymbol{q}, \boldsymbol{p}, N_{pr}, k, N_b, d, u_{t_p})$
$L \leftarrow \bigcup_{i \in [\![t - N_f; t + N_f]\!]} L_i$
$L \leftarrow$ elements from L with distance smaller than τ
`// the 3D transform requires a number of patch which is a power of 2`
$N_l \leftarrow$closest power of 2 small or equal than $min(sizeof L, N)$
if L *has more than* N_l *elements* **then**
  $L \leftarrow N_l$ best candidate from L
return L

σ	Method	Crowd	Park	Pedestrians	station	Sunflower	Touchdown	Tractor	Average
10	VBM3D (without)	35.52	34.59	40.65	38.38	39.81	39.01	36.82	37.83
	VBM3D (without,d=0)	35.72	35.00	40.13	39.34	40.18	39.01	36.94	38.05
	VBM3D (with)	35.61	34.94	**41.04**	39.79	41.75	**39.89**	38.43	38.78
	VBM3D (with,d=0)	**35.78**	**35.19**	40.61	**40.27**	**41.85**	39.75	**38.51**	**38.85**
20	VBM3D (without)	32.06	31.12	36.81	35.10	35.95	36.05	32.97	34.30
	VBM3D (without,d=0)	32.00	31.24	36.13	35.38	36.09	35.83	33.03	34.24
	VBM3D (with)	**32.17**	31.44	**37.32**	36.26	**38.02**	**36.91**	**34.58**	**35.24**
	VBM3D (with,d=0)	32.10	**31.49**	36.72	**36.32**	37.98	36.61	**34.58**	35.11
40	VBM3D (without)	28.39	27.64	32.62	31.80	32.31	33.35	29.38	30.78
	VBM3D (without,d=0)	28.29	27.65	32.24	31.79	32.33	33.11	29.40	30.69
	VBM3D (with)	**28.55**	**27.99**	**33.29**	**32.80**	**34.46**	**34.00**	**30.79**	**31.70**
	VBM3D (with,d=0)	28.45	27.98	32.94	32.75	34.40	33.79	30.76	31.58

Table 1.2: Comparison of the denoising quality with and without guiding with an optical flow. Guiding the search leads to much better results. It is also better in general to keep the additional regularization given by d.

1.3.2 Spatio-temporal patches

The patch similarity is determined based on the (squared) Euclidean distance between patches. Due to the noise, the Euclidean distance follows a non-central χ^2 distribution, with variance

$$\text{var}\left\{\frac{1}{m}\|\boldsymbol{q}_1 - \boldsymbol{q}_2\|^2\right\} = \frac{8\sigma^2}{m}\left(\sigma^2 + \frac{1}{m}\|\boldsymbol{p}_1 - \boldsymbol{p}_2\|^2\right),$$

where $\boldsymbol{q}_1, \boldsymbol{q}_2$ are the noisy versions of the patches $\boldsymbol{p}_1, \boldsymbol{p}_2$, and m denotes the number of pixels in the patch. The noise in the distance can be reduced by considering larger patches. However, increasing the spatial size of the patch also increases the distances between patches and reduces the likelihood of finding similar ones. Adding an additional temporal dimension in a spatio-temporal patch allows to increase the number of pixels in the patch, without increasing its spatial size. Due to the high redundancy of the video in the temporal dimension, increasing the temporal size of the patch causes a much lower increase in the patch distances. In practice, a spatio-temporal patch can be seen as a set of 2d patches from consecutive frames. Excepted in the case of motion compensated patched presented right after, these 2d patches are centered on the same spatial position in each frame.

When the motion is known or can be estimated, then it is natural to consider motion-compensated spatio-temporal patches (see for instance [MBFE12, BLM16]). Alternatively, rectangular spatio-temporal patches with no motion compensation have been also used [PE09, AM18a]. For more complex types of motion, using rectangular spatio-temporal patches will result in a larger variability in the set of nearest neighbors of a given patch, due to the fact that both the spatial texture and the motion pattern may vary. At least in principle, better results should be obtained using motion compensation. However, in practice, for higher levels of noise the bad quality of the estimated motion can undermine the final result.

The principle of BM3D has been applied to 3D patches with and without motion compensation. VBM4D, introduced in [MBFE12], uses motion-compensated spatio-temporal patches for video denoising (the "V" stands for video). The motion is estimated using block matching. BM4D uses cubic patches without motion compensation (of size $4 \times 4 \times 4$ or $5 \times 5 \times 5$), aiming at filtering volumetric images [MKEF13]. In [MKEF13] the authors compare the performance of both VBM4D and BM4D on video denoising concluding that VBM4D was the best video filtering strategy.

However, in [AM18a, AFM18a] it is shown that rectangular spatio-temporal patches with a temporal size of only two frames improve the denoising quality and still provide higher temporal consistency than a 2D patch. Based on those results, in this chapter we evaluate rectangular spatio-temporal patches of size $8 \times 8 \times 2$ in the first step and $7 \times 7 \times 2$ in the second (i.e we keep the spatial patch size).

In Table 1.3 we compare the quantitative results obtained by using spatial and spatio-temporal patches (denoted by "ST" in the table). We also consider the effect of guiding the patch search using an optical flow, indicated as "OF".

The first four columns of Figure 1.4 show results with/out motion-compensated search and spatio-temporal patches. From a qualitative point of view, using spatio-temporal patches provides better temporal consistency. In addition, the patch distance is more reliable since the number of pixels in the patches is doubled. This helps retrieve details and textures for regions with a simple motion (e.g. translational). For low noise levels, the effect of these 3d patches is mixed. While the results seem more consistent temporally, they are blurrier for sequences with complex motions. This explains the drop in PSNR observed for *pedestrians, sunflower* and *touchdown* between VBM3D and VBM3D ST for $\sigma = 10$. As the noise level increases this detail loss is out-weighted by the increased robustness to noise of the patch matching.

When spatio-temporal patches are used in conjunction with the optical-flow-guided search, their positive impact is magnified. Although the patches are not themselves motion-compensated, having a motion-compensated search region helps find better matches, even in sequences with more complex motion patterns. The motion-compensated search region also improves the temporal consistency, although to a lesser extent than the spatio-temporal patches. The best result, both in terms of PSNR and temporal consistency, is obtained when both strategies are used together (VBM3D ST+OF).

σ	Method	Crowd	Park	Pedestrians	Station	Sunflower	Touchdown	Tractor	average
10	VBM3D [DFE07]	35.65	34.75	40.83	38.93	40.49	39.04	37.01	38.10
	VBM3D (ours)	35.52	34.59	40.65	38.38	39.81	39.01	36.82	37.83
	VBM3D ST	**35.65**	34.66	40.41	38.55	39.65	38.91	36.90	37.82
	VBM3D OF	35.61	**34.94**	**41.04**	**39.79**	**41.75**	**39.89**	**38.43**	**38.78**
	VBM3D ST+OF	**35.74**	**35.04**	**41.01**	**40.41**	**41.91**	**39.98**	**38.71**	**38.97**
20	VBM3D [DFE07]	32.25	31.25	36.94	35.45	36.46	36.08	33.07	34.50
	VBM3D (ours)	32.06	31.12	36.81	35.10	35.95	36.05	32.97	34.30
	VBM3D ST	**32.39**	31.36	36.97	35.57	36.14	36.16	33.23	34.55
	VBM3D OF	32.17	**31.44**	37.32	36.26	**38.02**	**36.91**	34.58	**35.24**
	VBM3D ST+OF	**32.48**	**31.71**	**37.61**	**37.02**	**38.45**	**37.19**	**35.18**	**35.66**
40	VBM3D [DFE07]	28.65	27.68	32.81	32.02	32.65	33.52	29.41	30.96
	VBM3D (ours)	28.39	27.64	32.62	31.80	32.31	33.35	29.38	30.78
	VBM3D ST	**29.18**	28.13	33.35	32.50	32.70	33.65	29.66	31.31
	VBM3D OF	28.55	27.99	33.29	32.80	**34.46**	34.00	30.79	31.70
	VBM3D ST+OF	**29.30**	**28.50**	**34.21**	**33.68**	**35.06**	**34.47**	**31.46**	**32.38**

Table 1.3: Quantitative denoising results (PSNR and SSIM) for seven grayscale test sequences of size 960×540 from the *Derf's Test Media collection* for several variants of VBM3D. We highlighted the best performance in black and the second best in brown.

1.3.3 Multiscale video denoising

Multiscale approaches have shown to both reduce the residual noise but also improve the visual quality of the result. Indeed, most denoising algorithms work by processing local regions of the image/video (a patch, a group of patches, the receptive field of a neuron, etc). As a result, these methods fail to remove the lower frequencies of the noise. This results in smooth bumps mostly noticeable in large non-textured areas. Multiscale approaches are able to reduce this artifact by applying the denoising algorithm at different scales.

There are two main approaches for multi-scale denoising in the literature. The first one consists in modifying the denoising algorithm to consider several scales of the image/video. See for instance the multiscale version of the EPLL method [ZW11] proposed by Papyan and Elad [PE16]. Another approach proposed in [FPM17a, PMF17] considers the denoising algorithm as a black box, applying it as is at multiple scales. The result is then obtained by merging the results of each scale. This has the benefit that it can be applied to any denoising method, without any adaptation needed.

The multiscaler of [FPM17a] first creates a pyramid from the noisy input image $v^0 = v$ by applying a downscaling operator $\mathcal{D}$

$$v^{s+1} = \mathcal{D}(v^s), \text{ for } s = 0, ..., S-1. \tag{1.11}$$

At each scale, the denoising algorithm is applied and yields denoised images $\hat{u}^s$. These are then recomposed into a single multiscale output $\hat{u}_{\text{ms}}$. The recomposition is recursive. The recursion is

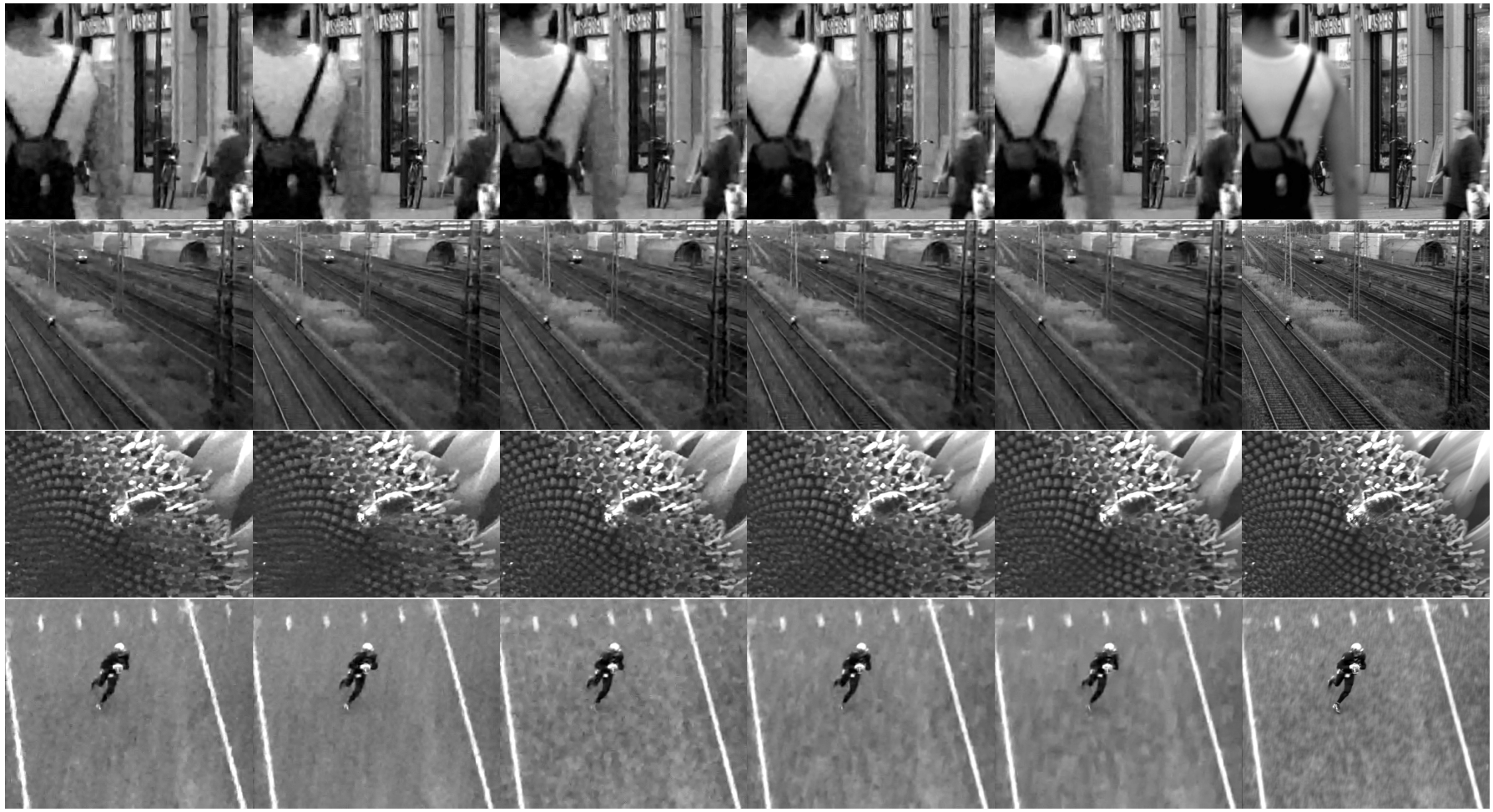


Figure 1.4: From left to right: result of VBM3D (our implementation with default parameters), VBM3D ST, VBM3D OF, VBM3D ST+OF, VBM3D ST+OF+MS (Lanczos multiscaler) and the original clean video. The noise level is $\sigma = 40$. Contrast has been linearly scaled for better visualization.

initialized at the coarsest level by defining $\hat{u}^S_{\text{ms}} = \hat{u}^S$, and then proceeds as follows:

$$\hat{u}^s_{\text{ms}} = \hat{u}^s - \mathcal{U}(\mathcal{L}(\mathcal{D}(\hat{u}^s), f_{\text{rec}})) + \mathcal{U}(\mathcal{L}(\hat{u}^{s+1}_{\text{ms}}, f_{\text{rec}})), \tag{1.12}$$

where U is the upscaling operator, and $\mathcal{L}(\,\cdot\,, f_{\text{rec}})$ is a low-pass filter with cutoff frequency parameterized by f_{rec} (f_{rec} does not depend on the scale). This equation substitutes the low frequencies of the single-scale result $\hat{u}^s$ with the multiscale solution $\hat{u}^{s+1}_{\text{ms}}$, computed using the coarser scales $s+1, ..., S$. The recomposition parameter f_{rec} determines which low frequencies are substituted. In one extreme, the low-pass filter lets all frequencies pass, in which case the whole coarse solution is used. At the other end, f_{rec} filters out all frequencies: the solution at the coarser level $\hat{u}^{s+1}_{\text{ms}}$ is discarded, and the output of the recomposition is the single scale denoising $\hat{u}^0$. In [FPM17a] it is found that the optimum is to filter out some of the high frequencies of the coarser level $\hat{u}^{s+1}_{\text{ms}}$. However, the exact amount depends on the denoiser.

To apply the multiscaler on a video, we apply it spatially, *i.e.* the temporal dimension is not downscaled. We first create a spatial pyramid of the entire video by creating a pyramid of each frame. We then denoise these videos, and recompose them by applying Eq. (1.12) to each frame. This is summarized in Algorithm 7.

Algorithm 7: : Multi-scale processing

input : A video v, a list of scales S, parameters for VBM3D, p
output : The denoised video $\hat{v}$

for *each scale* s *in* S **do**
  `// Compute the video` v^s `at the given scale`
  for *each frame* v_i *in* v **do**
    $v^s_i \leftarrow v_i$ at scale s
  `// Denoise the video` v^s
  $\hat{v}^s \leftarrow \text{VBM3D}(v^s, p)$
for *each frame index* i **do**
  $\hat{v}_i \leftarrow$ Combine the v^s_i for s in S
return $\hat{v}$

The parameters of the multiscaler are the number of scales, the downsampling ratio and the recomposition parameter f_{rec}. We shall set the downscaling ratio to 2 (the default), and try different values for the number of scales and the recomposition factor. There are different possibilities for the down/upscaling operators and the low-pass filter.

DCT pyramid. The DCT multiscaler uses a DCT pyramid which guarantees white Gaussian noise at all scales. The downscaling is performed by computing the DCT transform of the image and keeping only the quadrant of the image corresponding to the lowest frequencies. The upscaling is done by zero-padding in the DCT domain, and the low-pass filtering zeroes out the highest frequencies in the DCT domain. In this case f_{rec} represents the ratio of frequencies that are left: $f_{\text{rec}} = 1$ corresponds to an all-pass filter, where as $f_{\text{rec}} = 0$ filters out all the image.

Laplacian pyramid. The downscaling and upscaling operators samples the image using a Lanczos kernel:

$$k_a(x) = \begin{cases} \text{sinc}(x)\text{sinc}(x/a) & \text{if } |x| < a \\ 0 & \text{otherwise.} \end{cases} \tag{1.13}$$

We set $a = 3$. For the downscaling, to reduce aliasing, the image is downsampled using a scaled version of the kernel $k_3(\,\cdot\,/2)$ (as described in [Sch92]).[3] The low pass filter used is

[3] This corresponds to Matlab's `imresize` scaling function using the `lanczos3` interpolation kernel.

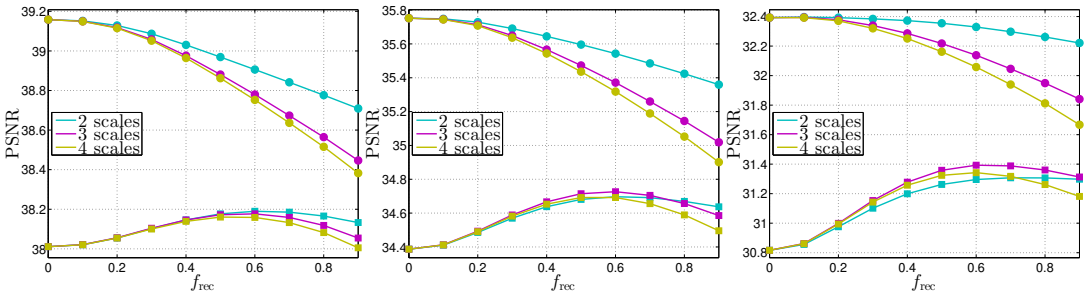


Figure 1.5: Effect of the DCT multiscaler for VBM3D without extensions (square markers) and VBM3D ST+OF (round markers). Each plot shows the average PSNR over our seven test sequences obtained when varying the recomposition factor f_{rec} for different number of scales. The single scale case correspond to $f_{\text{rec}} = 0$. From left to right, $\sigma = 10, 20, 40$. The multiscaler has a positive effect on the original VBM3D, but not on the improved VBM3D ST+OF.

the Gaussian filter of width f_{rec}. When $f_{\text{rec}} \to \infty$ we obtain the single scale output, and if $f_{\text{rec}} = 0$ no frequencies from the coarser solutions are discarded.

Figures 1.5 and 1.6 show plots of the average PSNR for our seven test sequences obtained with the two multiscalers varying the number of scales and the recomposition cutoff parameter f_{rec}. In each figure, we show the results of the multiscaler applied to the standard VBM3D, and to the one with statio-temporal patches and guided patch search (VBM3D ST+OF).

Both multiscalers have a positive impact when applied to the standard VBM3D. The gain can be up to 0.3dB for noise 10, 0.5dB for noise 20 and 0.8dB for noise 40. However, when applied to the VBM3D ST+OF version of the algorithm, the multiscaler does not improve the result. In fact, for noise 10 and 20, the best PSNR is attained by the single scale algorithm and the PSNR deteriorates as the cutoff frequency of the low-pass filter is increased (i.e. as more frequency components from the coarse solution are used).

The multiscaler achieves a better denoising of large objects with smooth textures by removing low-frequency noise left by the denoiser. This low-frequency noise is much stronger for the VBM3D denoiser than for the VBM3D ST+OF version. Hence, the improvement provided by the multiscaler is smaller for the latter. This also depends on the characteristics of the sequence. Frames with smooth objects occupying larger areas will benefit from the multiscaler. Yet, the multiscaler introduces artifacts penalizing the PSNR (particularly additional ringing for the DCT multiscaler). These artifacts are not temporally coherent and can therefore be quite noticeable.

Based on these plots we chose to select the Lanczos3 multiscaler. We use 2 scales and a recomposition factor $f_{\text{rec}} = 1$ when applied to VBM3D ST+OF (i.e. VBM3D ST+OF+MS) and 3 scales with recomposition factor $f_{\text{rec}} = 0.6$ when applied to the standard VBM3D (VBM3D MS). Table 1.4 shows the obtained PSNRs. The visual results of VBM3D ST+OF+MS are shown in Figure 1.4. The impact of the multiscaler can be noticed in the top row (results for *pedestrian*) as a reduction of the low-frequency noise in the smooth areas in the image. For the other sequences, since they are highly textured, the effect of the multiscaler is subtle. A careful examination reveals some texture loss.

1.4 Comparison with the state of the art and conclusion

In Table 1.5, we compare the PSNR of different recent denoising methods with VBM3D as well as with versions of VBM3D modified using different combinations of the improvements suggested in Sections 1.3.3, 1.3.2, 1.3.1. We included the patch-based methods SPTWO [BLM16] and VNLB [AM18a] and VNLnet [DEF+18], a convolutional neural network (CNN). SPTWO

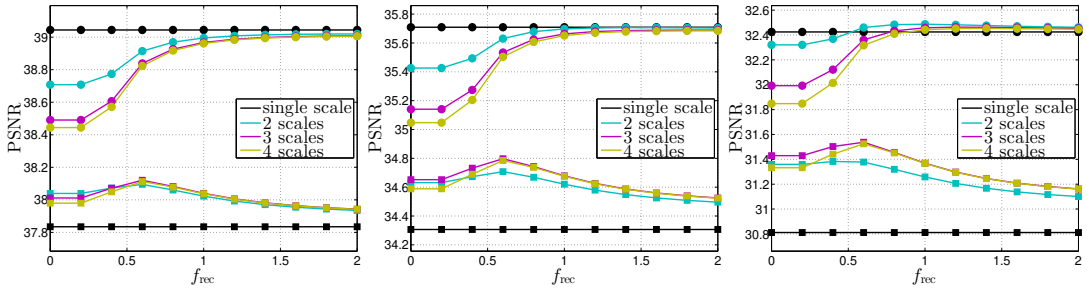


Figure 1.6: Effect of the Lanczos multiscaler for VBM3D without extensions (square markers) and VBM3D ST+OF (round markers). Each plot shows the average PSNR over our seven test sequences obtained when varying the recomposition factor f_{rec} for different number of scales. From left to right, $\sigma = 10, 20, 40$. The multiscaler has a positive effect on the original VBM3D, and on the improved VBM3D ST+OF for $\sigma = 40$.

σ	Method	Crowd	Park	Pedestrians	Station	Sunflower	Touchdown	Tractor	average
10	VBM3D (ours)	35.52	34.59	40.65	38.38	39.81	39.01	36.82	37.83
	VBM3D MS (DCT)	35.28	34.42	40.73	38.62	40.51	39.23	36.93	37.96
	VBM3D MS (Lanczos)	35.48	34.57	40.79	38.59	40.35	39.19	36.93	37.99
	VBM3D ST+OF	**35.74**	**35.04**	**41.01**	**40.41**	**41.91**	**39.98**	**38.71**	**38.97**
	VBM3D ST+OF+MS (DCT)	35.48	34.74	40.82	39.56	41.16	39.62	38.06	38.49
	VBM3D ST+OF+MS (Lanczos)	**35.72**	**35.01**	**41.05**	**40.34**	**41.88**	**39.97**	**38.66**	**38.95**
20	VBM3D (ours)	32.06	31.12	36.81	35.10	35.95	36.05	32.97	34.30
	VBM3D MS (DCT)	31.75	30.92	37.31	35.49	37.22	36.33	33.49	34.64
	VBM3D MS (Lanczos)	32.04	31.13	37.29	35.45	36.95	36.31	33.35	34.65
	VBM3D ST+OF	**32.48**	**31.71**	**37.61**	**37.02**	**38.45**	**37.19**	**35.18**	**35.66**
	VBM3D ST+OF+MS (DCT)	32.08	31.32	37.47	36.27	**37.71**	36.71	**34.51**	35.15
	VBM3D ST+OF+MS (Lanczos)	**32.46**	**31.68**	**37.74**	**36.99**	**38.45**	**37.16**	**35.18**	**35.67**
40	VBM3D (ours)	28.39	27.64	32.62	31.80	32.31	33.35	29.38	30.78
	VBM3D MS (DCT)	28.31	27.68	33.75	32.42	33.90	33.60	30.27	31.42
	VBM3D MS (Lanczos)	28.51	27.78	33.54	32.31	33.55	33.64	30.01	31.33
	VBM3D ST+OF	**29.30**	**28.50**	34.21	**33.68**	**35.06**	**34.47**	**31.46**	**32.38**
	VBM3D ST+OF+MS (DCT)	**28.90**	28.18	**34.25**	33.19	**34.41**	34.04	31.00	32.03
	VBM3D ST+OF+MS (Lanczos)	**29.30**	**28.51**	**34.46**	**33.70**	**35.06**	**34.50**	**31.56**	**32.44**

Table 1.4: Quantitative denoising results (PSNR and SSIM) for seven grayscale test sequences of size 960×540 from the *Derf's Test Media collection* for several variants of VBM3D. We used two scales and $f_{\text{rec}} = 1$ for ST+OF and three scales and $f_{\text{rec}} = 0.6$ for ST+OF+MS. We highlighted the best performance in black and the second best in brown.

and VNLB methods are based on grouping similar patches and filtering them on a transformed domain. However, instead of a pre-defined transform, an optimal transform is estimated for each group (the principal directions of the group of patches). While this gives good results, it requires computing an SVD of the matrix formed by the similar patches, resulting in very slow algorithms. VNLB groups 3D patches $10 \times 10 \times 2$, similar to our spatio-temporal patches. SPTWO instead uses long 3D spatio-temporal patches which are motion compensated, producing highly temporally consistent results. VNLnet applies a CNN to a stack of frames. The frames are all registered using block matching with a large block size. It produces good results with high temporal consistency except when the block matching fails to find good matches. Since it is implemented on GPU it runs much faster than VNLB and SPTWO, although it also has a high computational cost.

VBM3D combined with spatio-temporal patches and a patch-search guided with an optical flow gives very competitive results even compared to the latest state of the art, and especially for higher noises. When combined with the other improvements, it seems that multiscaling does not increase the quality of the denoising and can even degrade it in terms of PSNR. Visual examples are shown in Figures 1.4, where we compare results obtained with the original VBM3D with our implementation, using 3D patches and an optical flow to guide the search region.

In conclusion the proposed implementation of VBM3D and its variants, provide a video denoising framework represent an interesting trade-off between denoising quality and computational complexity.

σ	Method	Crowd	Park	Pedestrians	Station	Sunflower	Touchdown	Tractor	average
10	VBM3D [DFE07]	35.65	34.75	40.83	38.93	40.49	39.04	37.01	38.10
	BM4D [MKEF13]	35.84	34.45	41.15	40.23	40.97	39.78	37.33	38.54
	VBM4D [MBFE12]	36.05	35.31	40.61	40.85	41.88	39.79	37.73	38.88
	SPTWO [BLM16]	36.57	35.87	41.02	41.24	42.84	40.45	**38.92**	39.56
	VNLnet [DEF+18]	**37.00**	**36.39**	**41.96**	**42.44**	**43.76**	**41.05**	38.89	**40.21**
	VNLB [AM18a]	**37.24**	**36.48**	**42.23**	**42.14**	**43.70**	**41.23**	**40.20**	**40.57**
	VBM3D ST+OF+MS	35.72	35.01	41.05	40.34	41.88	39.97	38.66	38.95
20	VBM3D [DFE07]	32.25	31.25	36.94	35.45	36.46	36.08	33.07	34.50
	BM4D [MKEF13]	32.37	30.96	37.43	36.71	37.13	36.54	33.53	34.95
	VBM4D [MBFE12]	32.40	31.60	36.72	36.84	37.78	36.44	33.95	35.10
	SPTWO [BLM16]	32.94	32.35	37.01	38.09	38.83	**37.55**	35.15	35.99
	VNLnet [DEF+18]	**33.40**	**32.94**	**38.32**	**38.49**	**39.88**	37.11	**35.23**	**36.47**
	VNLB [AM18a]	**33.49**	**32.80**	**38.61**	**38.78**	**39.82**	**37.47**	**36.67**	**36.81**
	VBM3D ST+OF+MS	32.46	31.68	37.74	36.99	38.45	37.16	35.18	35.67
40	VBM3D [DFE07]	28.65	27.68	32.81	32.02	32.65	33.52	29.41	30.96
	BM4D [MKEF13]	29.10	27.82	33.44	32.98	33.06	33.68	29.84	31.42
	VBM4D [MBFE12]	28.72	27.99	32.62	32.93	33.66	33.68	30.20	31.40
	SPTWO [BLM16]	29.02	28.79	31.32	32.37	32.61	31.80	30.61	30.93
	VNLnet [DEF+18]	**29.69**	**28.29**	34.21	**33.96**	**35.12**	33.88	31.41	**32.51**
	VNLB [AM18a]	**29.88**	**29.28**	**34.68**	**34.65**	**35.44**	**34.18**	**32.58**	**32.95**
	VBM3D ST+OF+MS	29.30	28.51	**34.46**	33.70	35.06	**34.50**	**31.56**	32.44

Table 1.5: Quantitative denoising results (PSNR and SSIM) for seven grayscale test sequences of size 960×540 from the *Derf's Test Media collection* on several state-of-the-art video denoising algorithms. We highlighted the best performance in black and the second best in brown.

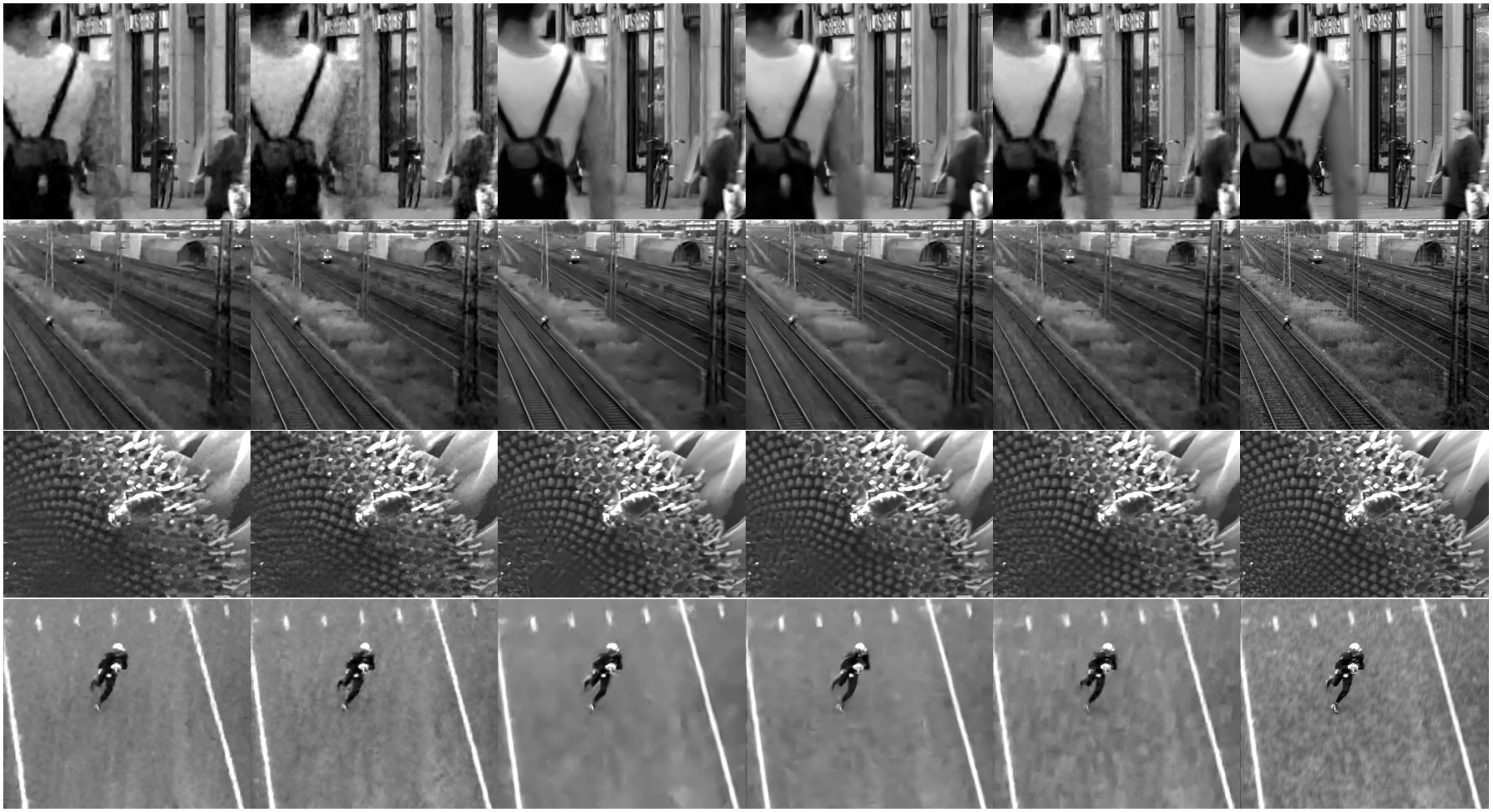


Figure 1.7: Comparison with state-of-the-art video denoising methods ($\sigma = 40$). From left to right: result of VBM3D [DFE07], VBM4D [MBFE12], VNLB [AM18a], VNLnet [DEF+18], VBM3D SF+OF+MS (Lanczos) and ground truth. Contrast has been linearly scaled for better visualization.

2 Global patch-search for denoising

With the increasing popularity of mobile imaging devices and the emergence of video surveillance, the need for fast and accurate denoising algorithms has also increased. Patch-based methods, which are currently state-of-the-art in image and video denoising, search for similar patches in the signal. This search is generally performed locally around each target patch for obvious complexity reasons. We propose here a new and efficient approximate patch search algorithm. It allows, for the first time, to evaluate the impact of a global search on video denoising performance. A global search is particularly justified in video denoising, where a strong temporal redundancy is often available. We first verify that the patches found by our new approximate search are far more concentrated than those obtained by exact local search, and are obtained in comparable time. To demonstrate the potential of the global search in video denoising, we take two patch-based image denoising algorithms and apply them to video. While with a classical local search their performance is poor, with the proposed global search they even improve the latest state-of-the-art video denoising methods. This work has been published in [EAM17].

2.1 Introduction

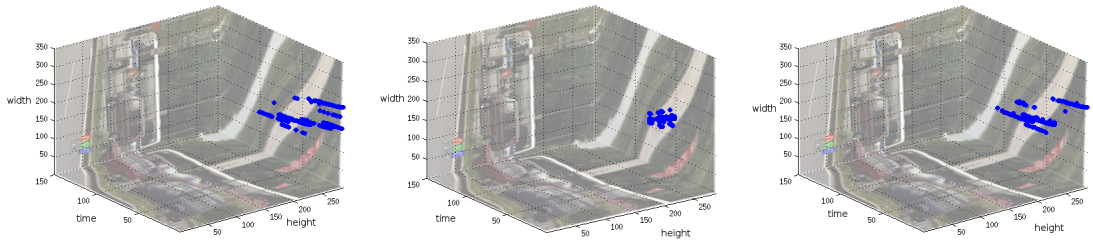


Figure 2.1: The plots show the position in the spatio-temporal video domain of the matches found for a sample patch query for different search methods. From left to right: the best matches found with a global exhaustive search, with a local exhaustive search in a window centered at the query, and with the VPLR search, the heuristic proposed in this chapter. Notice how the latter discovers the patch trajectories similar to those of the global exhaustive search.

Patch-based methods are among the state-of-the-art both in image denoising [DFKE07b, LBM13a, MBP+09] and video denoising [DFE07, MBFE12, LZD11, BLM16]. As we have seen in Chapter 1, these methods exploit the self-similarity of images and videos and filter together groups of similar patches which are then aggregated to create an estimate of the clean signal.

Most contributions in the area have focused on how to model and filter the groups of similar patches, but little attention has been given to how these groups are built. Typically similar patches are grouped by selecting a reference patch and searching exhaustively for the similar ones in a local 2D, or 3D for videos, neighborhood. The size of the search region is a parameter of the algorithm which trades off quality of the result for computational cost.

In the case of a single image, a local search region is justified by the fact that similar patches are likely to be close to each other in the image domain. Videos however, have an additional strong source of redundancy given by the temporal consistency. A patch is expected to have similar patches along its motion trajectory, even in distant frames. It seems intuitive that patch-based methods should benefit from this larger set of similar exemplars. Some methods estimate the motion in the video to tackle this problem. A motion compensated search window can track the patch trajectories for a certain number of frames [LF10, BLM16]. Nevertheless, the size of these search regions is still limited by the computational cost and the accumulation of errors in the estimated motion.

In this chapter we focus on the patch search. We present an efficient global approximate search technique and demonstrate its impact on video denoising. To that end we take two patch-based image denoising methods, namely BM3D [DFKE07b] and NL-Bayes [LBM13a] and adapt them to video (simply by searching similar patches in multiple frames instead of just the current one). We provide an extensive experimental evaluation in grayscale and color sequences. Our results show that substantial gains in performance are obtained by searching globally in the video sequence, indicating that video denoising still has significant room for improvement by using clever global search methods. In particular, the NL-Bayes method with global search outperforms state-of-the-art methods such as V-BM3D [DFE07] and V-BM4D [MBFE12] by a significant margin, and the recently proposed SPTWO [BLM16] by a lower margin.

In recent years several efficient techniques for approximate nearest neighbor search have been proposed, after the introduction of the PatchMatch algorithm by Barnes et al. [BSFG09a]. These methods compute a nearest neighbor field for patches located in a dense or semi-dense grid, and use heuristics that benefit from the overlap of adjacent patches in the grid. Most of these works focus on finding the nearest neighbor, but they can be extended to handle k nearest neighbors [BSGF10]. In practice k is kept small since even with these efficient techniques, computing a large number k of nearest neighbors for a dense grid of patches remains too costly.

In this chapter we focus on a significantly different problem: *compute a large number, namely k, of nearest neighbors for a single query patch*. By allowing independent queries, our technique is more flexible, and is straightforward to apply to patch-based denoising methods. In particular, this is useful for certain denoising algorithms which save computations by processing a sparse set of reference patches dynamically determined during the execution of the algorithm, and also makes parallelization easier.

The rest of the chapter is organized as follows: in §2.2 we describe briefly two state-of-the-art image denoising algorithms (NL-Bayes [LBM13a] and BM3D [DFKE07b]) which we selected to demonstrate our global search. In §2.3 we propose a new heuristic to accelerate approximate nearest neighbor search for patches in images and videos. A comparison with state-of-the-art methods is performed in §2.4. Concluding remarks are given in §2.5.

2.2 NL-Bayes and BM3D

BM3D and NL-Bayes are patch based methods which follow the same overall framework to denoise an image. For each patch in a set of patches to be denoised, they first search for similar patches inside the image. This search region is usually rectangular and centered on the query patch. The patches found during the search are then processed to compute an estimate of the

corresponding clean patches. The denoised patches are then aggregated to create a first (also called basic) estimate of the clean image. This process is then iterated once. In the second step, the basic estimate is used as a pilot for the patch search and the processing of the similar patches.

To test the proposed global search, we consider extensions to video of these algorithms by searching for similar 2D patches in a spatio-temporal volume in the video, as proposed in [DFE07], [AM15]. This framework is presented in Algorithm 8.

The main difference between BM3D and NL-Bayes algorithms lies in the processing of the set of similar patches. NL-Bayes learns a Gaussian *a priori* model for the set of patches and computes the patches as the *maximum a posteriori* estimate. BM3D stacks the patches in a 3D signal which is denoised using shrinkage on a transformed domain. A more detailed presentation of BM3D is available in Chapter 4.

We use a slight modification of video NL-Bayes [AM15] which caps the rank of the patch covariance matrices for the groups of similar patches. This improves both performance and speed of this algorithm, and adds a rank parameter r to each step.

We modified the search (corresponding to steps 2 and 6 in Algorithm 8) by considering three different approaches: a *local* approach which uses the local search in a spatio-temporal volume centered at the reference patch for both denoising iterations; a *global* approach which searches in the full video volume for both steps of the algorithm; and *mixed* approach which uses the local search in the first step and the global in the second iteration (step 6 in the pseudocode). The global patch search heuristic is presented in Section 2.3.

Algorithm 8: Image/video denoising framework

input : Noisy image/video v, noise level σ
output : Estimate of noiseless image/video $\hat{v}$

1 **for** *all patch* $\boldsymbol{q}$ *in* v **do**
2 **Retrieve** n **nearest neighbors to** $\boldsymbol{q}$
3 Process the set of similar patches and compute a denoised estimate $\boldsymbol{q}'$ of $\boldsymbol{q}$
4 Aggregate estimated patches on v' to compute the basic estimate
5 **for** *all patch* $\boldsymbol{q}$ *in* v' **do**
6 **Retrieve** n **nearest neighbors to** $\boldsymbol{q}$
7 Process the set of similar patches and compute a denoised estimate $\hat{\boldsymbol{q}}$ of $\boldsymbol{q}$
8 Aggregate estimated patches on $\hat{v}$ to compute the final estimate
9 **return** $\hat{v}$

In the Section 2.4 we shall compare these approaches among them and also with other video extensions of BM3D and NL-Bayes which use sophisticated local patch search regions, V-BM3D (presented in Chapter 1) and SPTWO. V-BM4D [MBFE12] is an extension of V-BM3D.

SPTWO [BLM16] is based on NL-Bayes but performs a more elaborate search. First, the optical flow towards the adjacent six frames is computed and used to warp them to the reference frame. A $s \times s \times 13$ spatio-temporal patch is then associated to each $s \times s$ patch in the reference frame by extending its temporal dimension on the volume defined by the warped frames. Then, k patches closest to the reference are searched for in a local neighborhood. The final set of matches is given by the $13k$ 2D slices from the newly found extended patches (some of them might be discarded by an occlusion detection step). The usage of these extended patches reduces the noise in the distance while still keeping a small spatial patch (the patch spatial size is $s = 5$). Furthermore, the patches are warped by the optical flow. This is useful in cases where the motion is not translational. On the downside, this method relies heavily on the optical flow.

2.3 Heuristics for global patch search

There are already a number of efficient patch search techniques which exploit the so-called image coherency: Neighboring query patches, since they overlap, have high chances of having neighboring matches in the database image. Thus knowing the position of a good match for a patch helps in determining good matches for its neighbors. This idea was first applied by [BSFG09a] to compute a nearest neighbor field (NNF), assigning the closest k patches to each patch in the image. The original algorithm was presented for images but can easily be extended to video. More recent works reported significant improvements (between one and two orders of magnitude) by combining PatchMatch with more classic search data structures such as partition trees (more specifically KD-trees) [OA12, HS12] and locality sensitive hashing [KA11, BZL+15].

All these methods compute a dense k-NNF, typically for a small k. The reason is that when k is large (e.g. $k > 20$) computing a dense NNF is too costly. In such cases it is preferable (if the application allows to) to compute the k nearest neighbors for a small set of patches. In particular, for image/video denoising, a common speed-up strategy of patch-based methods is to reduce the number of query patches. For instance, in [DFKE07b] the query patches form a regular subgrid, and in [LBM13a] the query patches are irregularly located and are determined during the evolution of the algorithm. This is why we need a method that can compute nearest patches for a set of selected patches without having to do it for all patches of the image/video. There is a vast literature on data structures for nearest neighbor search on generic metric spaces or vector spaces; but these classical tools do not exploit the image coherency. In this section we briefly review one of such approaches, namely partition trees, and show a simple yet effective modification for a fast approximate k-nearest neighbor search of image patches.

2.3.1 Partition Trees

A partition tree is an inductive data structure encoding the position of a set of n points in $\mathbb{R}^d$ (*the database*). Once the partition tree has been built it is used to search for the nearest neighbors of a point (the *query*). Nodes in the tree can either be leaves (also called bins) containing a maximum number of elements, or a split value between two subtrees: the "left" subtree and the "right" subtree. A partition tree splits recursively the data space by applying a simple split at each node (with the exception of the leaves). At each splitting operation, the set of elements in the current subtree is split in two equally sized subsets which are then used to construct the child subtrees. The construction of the tree is fully specified by a *split value function* and a *split function*. The split value function assigns a split value to a set of elements, whereas the split function assigns one of two groups to an element.

A partition tree can be directly used to search for the *exact* k-nearest neighbors with expected complexity of $O(\log(n))$, where n is the number of points. In practice when the dimensionality of the elements is too large, its performance drops and becomes comparable to a linear search, this problem was shown with image patches by Kumar et al. in [KZN08]. This is indeed the case for image and video patches, so we shall rule out exact search and settle with the so-called *first bin heuristic*: The candidates for the k-nearest neighbors are taken only from the bin of the query patch. On its own, the *first bin heuristic* does not suffice to provide good quality matches, but this will be solved by combining it with other heuristics that exploit the fact that patches lie on images. This allows to be much faster than other approximative search such as FLANN [ML09] for the same quality of nearest neighbors.

By randomizing the construction of these trees, forests can be built. It is then possible to perform independent parallel queries with each tree and select the best elements, therefore improving the quality of the elements retrieved using the *first bin* search. Since the queries can be done in parallel, this improvement does not take more time than the basic tree search.

2.3.2 Partition Tree Search With Local Refinement

We propose to use the first bin search in a partition tree (or a forest) to obtain a first set of $m \approx k$ initial candidates matches, and refine this set of candidates as follows: For each of the elements of the candidate list, we search in a small 2d local region in the image centered at the candidate. We call the resulting strategy Partition Tree with Local Refinement (PTRL). The complete pseudocode for this technique is presented in Algorithm 9. This refinement is inspired by the local refinement of PatchMatch that will be studied in Chapter 10.

Note that the proposed approximate search heuristic can in principle be applied in conjunction to other data structures for nearest neighbor search such as those based on hashing [AI06, AINR14]. We do not pursue this in the present work.

Algorithm 9: PTLR search heuristic

input : v an input video, $\mathcal{F}$ a Partition tree forest constructed with the patches from v, p a request patch from v, $\kappa \times \kappa$ the size of local search region

output : A list of matches for p

Retrieve the list $\{\varphi_1, \ldots, \varphi_k\}$ of k best matches from the forest $\mathcal{F}$ using the retrieval algorithm from a partition tree forest

for $i = 1$ *to* k **do**

  Perform an exhaustive search in a local 2d region of size $\kappa \times \kappa$ centered on φ_i for better matches

return the list $\{\varphi'_1, \ldots, \varphi'_k\}$ of k best matches after the update using the PTLR search

2.3.3 Search Parameters

The choice of the partition tree has a strong impact on the performance. The most common partition tree is the KD-tree [Ben75]. However, it has been shown that VP-trees [Yia93] produce better results when working with image patches [KZN08].

The VP-tree is characterized by the split value being a hyperball and the split function being the indicator function of this ball. The VP-tree splits the data set according to the distance of each point to a *vantage point*. The vantage point is one of the data points, chosen according to some criteria. Forests of VP-trees where the vantage point is chosen at random have been shown to have a good retrieval power [OD+13].

For our experiments, we used a forest of four VP-trees, randomized as in [OD+13]. We set the size of the bins to $2n$, where n is the number of nearest neighbors of the query. The trees are constructed using all patches in the video. For the query, the local refinement area is of size $8 \times 8 \times 3$. We found these parameters to give a reasonable trade-off between computational cost and search accuracy. In the following, we will use the abbreviation VP with local refinement (VPLR) instead of partition tree with local refinement (PTLR) to remind that the partition tree is specifically a VP-tree.

In Figure 2.2 we compare the results obtained with the local search, the "first bin" search, and the proposed VPLR search for the classical test sequence *bus*[1]. The plot shows the Euclidean distances to the $n = 300$ nearest neighbors averaged over 1000 query patches randomly chosen in the same video. The patches are of size $s \times s$ with $s = 10$, RGB (thus their dimensionality is 300) and their distance is rescaled between 0 and 1 as $d(\boldsymbol{p}, \boldsymbol{q}) = \|\boldsymbol{p} - \boldsymbol{q}\|/(255\sqrt{3}s)$.

The first conclusion is that the local exhaustive search finds worse matches than the approximate global search methods (the search window of the local search is of size $45 \times 45 \times 5$). The best method is the VPLR search, performing much better than the basic VP-forest "first bin" approx-

[1] https://media.xiph.org/video/derf/

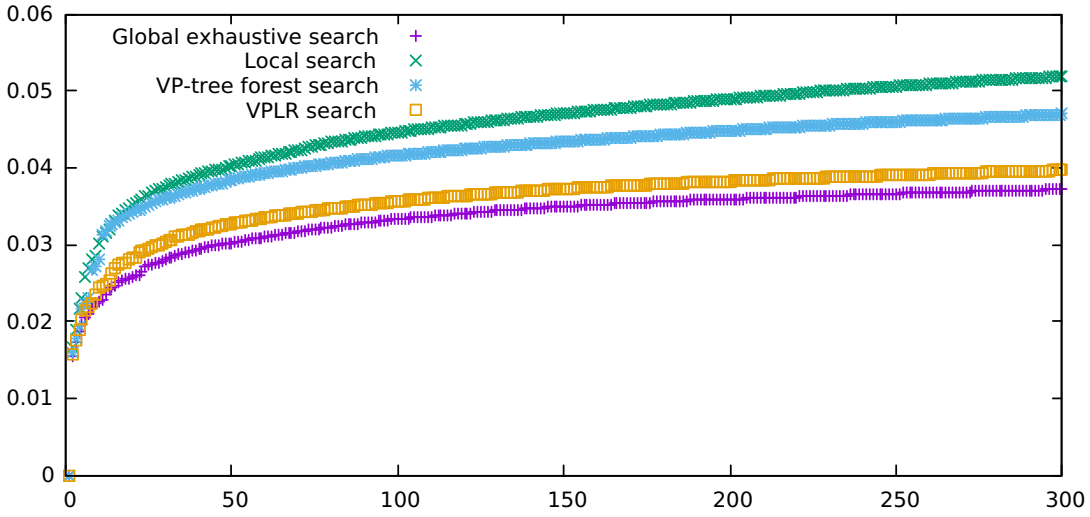


Figure 2.2: Comparison of the quality of the matches. The plots show the normalized distance to the ith nearest neighbor $i = 1, ..., 300$, averaged over 1000 query patches sampled randomly in the *bus* video.

Method	σ	n_1	n_2	r_1	r_2
NLB-local grayscale	all	160	$40 + \frac{4}{7}(\sigma - 6)$	$\max\{8, 48 - \frac{16}{21}(\sigma - 6)\}$	$\max\{8, 16 - \frac{4}{9}(\sigma - 6)\}$
NLB-global grayscale	$\leqslant 10$	100	60	16	16
	> 10	100	60	16	8
NLB-local/global color	all	350	150	full rank	$40 - (\sigma - 10)/3$
BM3D-mix	$\leqslant 20$	$4 \times 2^{\sigma/10}$	32	-	-
	> 20	32	32	-	-
BM3D-global	all	32	32	-	-

Table 2.1: Parameters used for the different algorithms.

imate search. This result shows the effectiveness of the proposed search heuristic. We computed the corresponding plot on the other videos and always found the same qualitative behavior.

It is also interesting to visualize the position of the matches found by each method in the spatio-temporal domain of the video. Figure 2.1 presents the position of the nearest patches found for a specific query in the bus video. Note how the matches found by searching globally are organized in trajectories.

For the same parameters, the number of distance computations for each method is 10125 and around 20000 for respectively the local search and the VPLR search for the first step of the algorithm (an equivalent local search region would be close to $63 \times 63 \times 5$).

2.4 Experimental results

We evaluated the effect of the global search on the BM3D and NL-Bayes image denoising algorithms. Since the source code of BM3D is not public we use the implementation available in [Leb12]. We adapted it to process image sequences and modify only the patch search as explained in §2.2. For NL-Bayes (referred as NLB in the following), we built our implementation upon [LBM13b] , and modified to limit the rank of the *a priori* covariance matrix (see §2.2).

For each method we considered the three versions depending on the type of search used in each step: the *local* (exhaustive in a small spatio-temporal volume) and *global* (approximative in the entire video) versions use the corresponding search in both denoising steps, and an additional *mixed* approach, which uses the local search in the first step and the global one in the second. The reason for this will be explained later.

We compared these methods against V-BM3D [DFE07], V-BM4D [MBFE12] and SPTWO [BLM16], which represent the current state-of-the-art in video denoising.

Regarding the parameters, we considered 2D patches of size 10×10 for NL-Bayes and 8×8 for BM3D. For the local search we used a window of size $45 \times 45 \times 5$ for NL-Bayes and $32 \times 32 \times 5$ for BM3D.

The remaining parameters for NL-Bayes are the number of similar patches used in each step n_1, n_2 and the maximum ranks of the *a priori* covariance matrix r_1, r_2. For BM3D we also needed to specify n_1, n_2, in addition to the hard threshold in the first step λ_1 as well as the distance threshold in both steps, τ_1 and τ_2. The values for λ_1, τ_1 and τ_2 are the same as the ones in VBM3D. The rest of the parameters were tuned by optimizing the PSNR on a training set consisting of short videos. The optimal parameters depend on the noise level. Table 2.1 synthesizes the different parameters as a function of the noise.

These patches are somewhat larger than the typical sizes used by patch-based methods. The reason for this is that the global search is more sensitive to the noise in the patch distance. Consider for example a noisy flat image. Since the image is flat, the closest neighbors to a patch are the ones with the closest noise pattern. If enough nearest neighbors share the same noise pattern, it will be interpreted as a signal component and will not be filtered out. The global search increases the probability of finding a large number of matches with a similar noise pattern (overfitting to this specific noise pattern). In practice, this becomes an issue for patches in homogeneous regions and high noise values. This problem can be mitigated by involving a larger number of similar patches. An interesting aspect of global search is that it is still possible to find many similar patches even for large patch sizes.

Our quantitative comparison criterion was the PSNR. We first evaluated the effect of the global search for NL-Bayes and BM3D. Tables 2.2 and 2.3 show the gain in PSNR with a global search, on grayscale and color sequences.[2]. In almost all cases the global search performed significantly better than the local for NL-Bayes. This is also true, but to a minor extent for BM3D. For NL-Bayes the highest gain was obtained using the global search in both steps of the denoising algorithm: The average gain between NLB-global and NLB-local is of around 1dB for grayscale sequences and of 1.5dB for color sequences. This gain is consistent across the different noise levels we used in our tests. For BM3D the best alternative is BM3D-mix, which uses the global search only in the second step. The performance of BM3D-global is superior for $\sigma = 10$, but drops severely when the noise increases, becoming comparable or even worse than BM3D-local for the highest levels of noise.

A possible reason for this is that BM3D uses a much smaller number of similar patches n than NL-Bayes. As explained before, for patches with low SNR, the global search increases the risk of finding a set of nearest neighbors sharing a similar noise pattern, particularly for small n. Note also that the performance of BM3D is worse than that of NL-Bayes, and in most cases worse than V-BM3D (see Table 2.4). This is because our version of BM3D is an adaptation to video of the one published in [Leb12]. In particular, the search strategies of our BM3D-local and V-BM3D differ, V-BM3D uses the predictive block matching described in §2.2 that is optimized to search in a large temporal window while our search is very local both in space and in time.

[2]The color sequences are from https://media.xiph.org/video/derf/. The grayscale sequences are from http://www.cs.tut.fi/~foi/GCF-BM3D/, except for *football* and *mobile*, which have been obtained by averaging the channels from the corresponding RGB sequences.

σ	Method	Bus	Fore.	Sales.	Tennis	Foot.	Mobi.	Ave.
10	NLB-local	34.85	36.33	35.87	33.94	35.29	34.29	35.10
	NLB-mix	+0.54	+0.36	+0.90	+0.42	+0.06	+1.30	+0.60
	NLB-global	+0.94	+0.50	+2.01	+0.64	-0.02	+1.60	+0.95
	BM3D-local	34.25	35.77	35.79	33.55	35.15	32.97	34.58
	BM3D-mix	+0.15	+0.55	+1.08	=0.09	-0.08	+0.81	+0.43
	BM3D-global	+0.09	+0.92	+1.66	-0.03	-0.19	+1.54	+0.67
20	NLB-local	30.75	32.59	32.06	30.12	31.36	29.80	31.11
	NLB-mix	+0.53	+0.62	+1.15	+0.50	+0.11	+1.91	+0.80
	NLB-global	+0.70	+0.69	+1.96	+0.70	-0.05	+2.46	+1.08
	BM3D-local	30.28	32.17	31.84	29.90	31.23	29.02	30.74
	BM3D-mix	+0.17	+0.52	+0.74	+0.19	+0.05	+0.81	+0.41
	BM3D-global	+0.05	+0.50	+0.78	+0.13	-0.14	+1.52	+0.47
30	NLB-local	28.46	30.53	29.86	27.99	29.26	26.95	28.84
	NLB-mix	+0.46	+0.06	+0.55	+0.63	-0.14	+2.24	+0.63
	NLB-global	+0.41	+0.04	+0.84	+0.73	-0.33	+2.68	+0.73
	BM3D-local	28.06	29.92	29.38	28.12	29.18	26.47	28.52
	BM3D-mix	+0.23	+0.48	+0.63	+0.45	+0.15	+0.65	+0.43
	BM3D-global	+0.00	+0.07	+0.38	+0.34	-0.19	+0.92	+0.25

Table 2.2: Comparison between search strategies on grayscale sequences. For each sequence and noise level, we show the PSNR obtained with the local search, and the difference in PSNR between each global search strategy and the local search.

We compared the performance of NLB-local and NLB-global with the state-of-the-art methods V-BM3D [DFE07], V-BM4D [MBFE12] for grayscale (Table 2.4) and color (Table 2.5) videos. The sequences used in these tables were the same as in Tables 2.2 and 2.3. The results of V-BM3D[3] and V-BM4D were computed using the authors' implementation.[4] For grayscale sequences and $\sigma = 10$, NLB-global has the best performance. On average NLB-global has a PSNR .78dB higher than V-BM4D. When the noise increases, the gap between NLB-global and the V-BMxD methods closes. For most sequences, better results can be obtained with NLB using a larger patch for higher noise levels. This suggest that the problem comes from the distance estimation in these high noise cases.

We also include a comparison with SPTWO in Table 2.6. We computed the result of our algorithm for some of the sequences used in [BLM16]. Note that the sequences in Table 2.6 have 30 frames and that the values shown correspond to the PSNR of the central frame of the sequence. The results depends largely on the sequence. In particular SPTWO performs better in *tennis*, and *bus*. The fact that the *bus* sequence has a very fast motion which can be easily estimated might explain the performance gap in favor of motion estimation on this sequence.

Two examples of denoised results are presented in Figures 2.5 and 2.6. Comparing the different results of denoising, we can see that the VPLR search allows a better detail reconstruction. In

[3] For V-BM3D we show only grayscale results since there are no Linux binaries for the color version of V-BM3D.
[4] http://www.cs.tut.fi/~foi/GCF-BM3D/

σ	Method	Bus	City	Cont.	Mobile	Tennis	Fore.	Coast.	Ave.
10	NLB-local	36.47	37.35	38.29	34.76	35.30	38.34	36.60	36.73
	NLB-mix	0.64	1.37	1.16	1.46	0.74	1.18	0.52	1.01
	NLB-global	0.83	2.06	1.72	2.25	0.97	1.64	0.75	1.46
	BM3D-local	35.57	36.50	37.26	33.59	34.61	37.60	35.73	35.84
	BM3D-mix	0.22	0.73	1.08	0.44	0.40	0.57	0.22	0.52
	BM3D-global	0.12	1.02	1.44	0.76	0.19	0.70	0.21	0.63
20	NLB-local	32.42	33.31	34.47	30.74	31.52	35.09	32.69	32.89
	NLB-mix	0.73	1.76	1.77	1.58	0.67	1.04	0.67	1.17
	NLB-global	0.90	2.34	2.34	2.58	0.84	1.37	0.86	1.60
	BM3D-local	31.72	32.75	33.68	29.75	30.87	34.39	31.99	32.16
	BM3D-mix	0.21	0.71	1.30	0.55	0.24	0.72	0.26	0.57
	BM3D-global	-0.01	0.43	1.68	1.21	-0.04	0.62	0.17	0.58
40	NLB-local	28.52	29.24	30.79	26.62	28.25	32.18	29.09	29.24
	NLB-mix	0.71	1.32	2.33	1.93	0.64	0.83	0.75	1.22
	NLB-global	0.73	1.28	2.78	2.93	0.59	0.82	0.82	1.42
	BM3D-local	27.89	28.45	30.23	26.07	27.62	31.04	28.34	28.52
	BM3D-mix	0.35	0.48	1.49	0.80	0.53	1.00	0.47	0.73
	BM3D-global	-0.13	-0.54	1.79	1.28	0.25	0.54	0.11	0.47

Table 2.3: Comparison between search strategies on color sequences. For each sequence and noise level, we show the PSNR obtained with the local search, and the difference in PSNR between each global search strategy and the local search.

σ	Method	Bus	Fore.	Sales.	Tennis	Foot.	Mobi.	Ave.
10	V-BM3D*	33.32	36.02	37.21	34.68	34.82	34.09	35.02
	V-BM4D*	33.85	36.36	37.48	**34.78**	34.95	34.11	35.26
	NLB-local	34.85	36.33	35.87	33.94	**35.29**	34.29	35.09
	NLB-global	**35.79**	**36.83**	**37.88**	34.58	35.27	**35.89**	**36.04**
20	V-BM3D*	29.57	32.87	**34.04**	**31.20**	31.04	30.35	31.51
	V-BM4D*	30.00	33.11	33.46	30.70	31.06	30.49	31.47
	NLB-local	30.75	32.59	32.06	30.12	**31.36**	29.80	31.11
	NLB-global	**31.45**	**33.28**	**34.02**	30.82	31.31	**32.26**	**32.19**
30	V-BM3D*	27.59	30.85	**31.68**	**29.22**	29.04	27.85	29.37
	V-BM4D*	27.96	**31.06**	31.02	28.74	28.98	27.99	29.29
	NLB-local	28.46	30.53	29.86	27.99	**29.26**	26.95	28.84
	NLB-global	**28.87**	30.57	30.70	28.72	28.93	**29.63**	**29.57**

Table 2.4: Comparison with V-BM3D and V-BM4D on grayscale sequences. PSNR of the full sequence. See text for details. Results with a star were computed using the binary provided by the author.

σ	Method	Bus	City	Cont.	Mobile	Tennis	Fore.	Coast.	Ave.
10	V-BM4D-mp*	35.39	37.14	38.78	34.18	35.91	37.95	36.05	36.49
	NLB-local	36.47	37.35	38.29	34.76	35.30	38.34	36.60	36.73
	NLB-global	**37.30**	**39.41**	**40.01**	**37.01**	**36.27**	**39.98**	**37.35**	**38.19**
20	V-BM4D-mp*	31.35	33.41	34.94	30.47	31.99	34.53	32.14	32.69
	NLB-local	32.42	33.31	34.47	30.74	31.52	35.09	32.69	32.89
	NLB-global	**33.32**	**35.65**	**36.81**	**33.32**	**32.36**	**36.46**	**33.55**	**34.50**
30	V-BM4D-mp*	29.04	31.04	32.63	28.35	29.73	32.54	29.97	30.47
	NLB-local	30.10	30.92	32.32	28.33	29.53	33.37	30.55	30.73
	NLB-global	**30.92**	**32.79**	**34.97**	**31.17**	**30.24**	**34.42**	**31.38**	**32.27**
40	V-BM4D-mp*	27.44	29.31	30.94	26.79	28.15	31.08	28.49	28.89
	NLB-local	28.52	29.24	30.79	26.62	28.25	32.18	29.09	29.24
	NLB-global	**29.25**	**30.52**	**33.57**	**29.55**	**28.84**	**33.00**	**29.91**	**30.66**
50	V-BM4D-mp*	26.24	27.97	29.60	25.52	27.03	29.90	27.34	27.66
	NLB-local	27.33	27.98	29.59	25.29	27.31	31.22	27.99	28.10
	NLB-global	**27.99**	**28.77**	**32.39**	**28.22**	**27.86**	**31.87**	**28.79**	**29.41**

Table 2.5: Comparison with V-BM4D on color sequences. PSNR of the full sequence. See text for details. Results with a star were computed using the binary provided by the author.

σ	Method	Bus	Tennis	Salesman	Bike	Average
10	SPTWO	**36.07**	**34.69**	36.38	36.74	35.97
	NLB-local	34.89	32.86	35.92	37.10	35.19
	NLB-global	35.60	34.31	**37.10**	**38.30**	**36.33**
20	SPTWO	**32.24**	**30.59**	32.95	33.01	32.20
	NLB-local	30.75	28.23	32.02	33.39	31.10
	NLB-global	31.02	28.71	**34.45**	**35.56**	**32.44**
30	SPTWO	**30.05**	**27.48**	30.95	31.62	30.02
	NLB-local	28.48	26.24	29.92	31.17	28.95
	NLB-global	28.87	26.63	**31.49**	**33.28**	**30.07**

Table 2.6: Comparison with SPTWO. As in [BLM16], only the first 30 frames of the sequence are considered, and the shown PSNRs corresponding to the frame 15 of each sequences (indexing starts with 1). The values in each cell correspond to SPTWO, NLB-local and NLB-global.

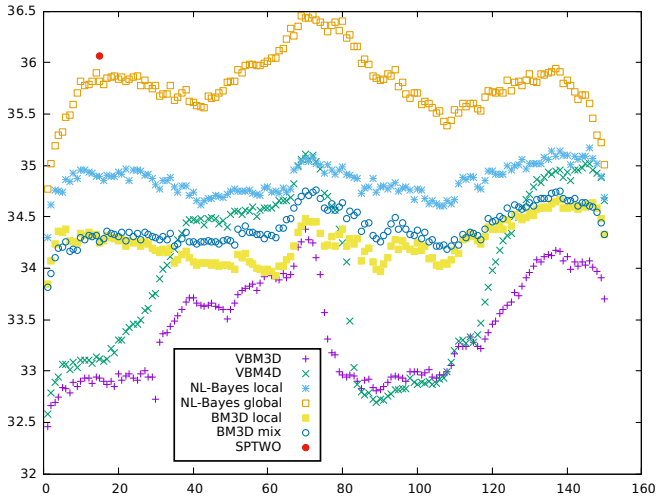


Figure 2.3: Comparison of the PSNR frame by frame for different methods on the grayscale bus sequence with noise 10.

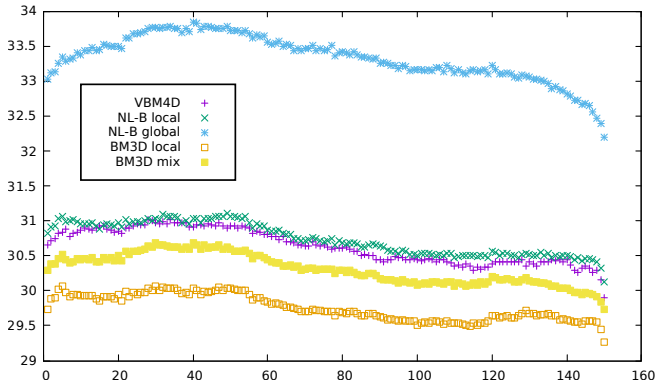


Figure 2.4: Comparison of the PSNR frame by frame for different methods on color mobile sequence with noise 20.

particular, in Figure 2.5 the numbers of the calendar do not show a blur around them compared to the other methods based on a local search. For the example from *grayscale bus*, the improvements can be seen mostly for the woman inside the bus, who is more distinct with the global search than with the local search; but also on the ad where most details of the singer's face are better reconstructed.

In section 2.3.3, we briefly discussed the computation complexity (in number of distance computation) of each search method per query. When these searches are integrated into the denoising algorithm, the full computation time (including the construction of the VP-tree) is of the same order of magnitude than the one when using the local search. Nevertheless, NL-Bayes based methods are reasonably slower than V-BM3D and V-BM4D when using the "normal profile".

2.5 Conclusions

We studied the performance gain obtained by expanding the local patch search into a global one for patch-based video denoising algorithm. To the best of our knowledge, this is the first time that denoising results using global patch search were reported in videos with hundreds of frames. With the global search the patches found can follow long trajectories in the video, thus fully benefiting

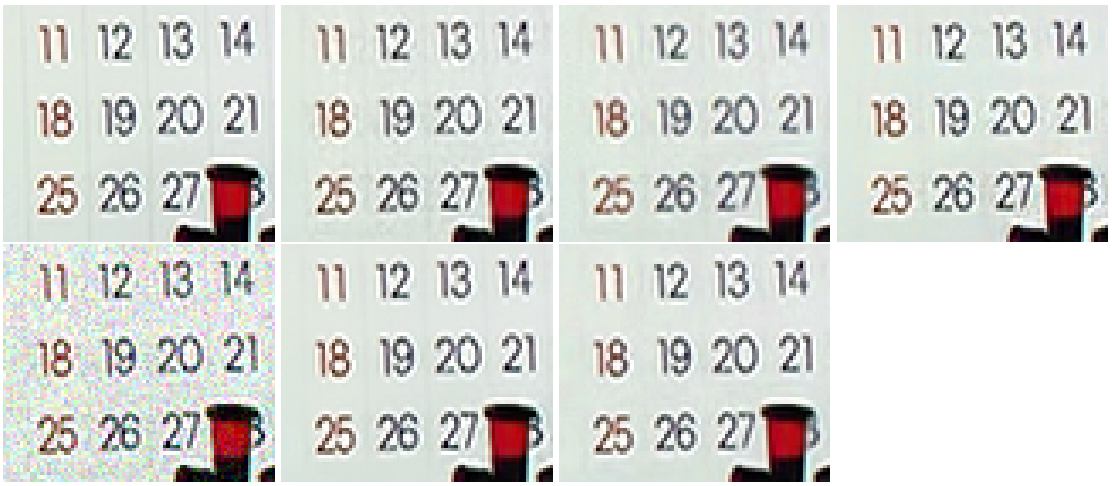


Figure 2.5: Results of denoising for *mobile* (zoom on frame 37, noise 20). Top: ground truth, NL-Bayes local, BM3D local, VBM4D; bottom: Noisy, NL-Bayes global and BM3D mix.

Figure 2.6: Results of denoising for grayscale *bus* (zoom on frame 70, noise 20). Top: ground truth, NL-Bayes local, BM3D local, VBM4D; bottom: Noisy, NL-Bayes global, BM3D mix and VBM3D.

from the temporal redundancy of videos.

Our analysis of the most common patch search algorithms showed that an approach based on a global tree structure, more specifically based on a VP-tree, performed very well compared to the local search. Exact global search in the VP-tree is still too costly for the denoising application, which is why we proposed a simple heuristic for efficient approximate search, the VPLR search (VP-tree search with local refinement).

We then applied it to extend BM3D and NL-Bayes, two image denoising algorithms, to video. We obtained a significant boost on the denoising performance. This performance boost is only slightly more costly than a local exhaustive search, including the time spent building the tree thanks to an easy parallelization.

Latest contributions in video denoising advocate for the use of 3D patches as a mechanism to impose temporal consistency in the video [PE09, LF10, MBFE12]. Yet, in this chapter we showed that state-of-the-art results can be obtained with 2D patches, using global search. The results obtained are visually better frame-by-frame, but can suffer from a flickering artifact due to the lack of temporal consistency. This is most noticeable for higher values of noise. Ongoing work focuses on extending the current results to 3D patches and video specific algorithms. One of the

current limiting factors associated to the global search is that it increases the risk of matching the noise pattern for patches with low SNR. We were able to alleviate this problem in most cases by using large 2D patches, but this causes problems with random, low-contrasted textures which are better denoised with small patches. 3D patches can reduce the spatial patch size while still keeping accurate distances (same dimension than the 2D patches), and therefore be more appropriate for these types of textures.

The proposed heuristics for approximate global patch search are not limited to the denoising application and are useful for other applications requiring a large number of nearest neighbors. An example of application to anomaly detection is presented in Chapter 9.

3 A recursive video denoising method: NL-Kalman

In this chapter we propose a new recursive video denoising method with high performance. The method is recursive and uses only the current frame and the previous denoised one. It considers the video as a set of overlapping temporal patch trajectories. Following a Bayesian approach each trajectory is modeled as linear dynamic Gaussian model and denoised by a Kalman filter. To estimate its parameters, similar patches are grouped and their trajectories are considered as sharing the same model parameters. The filtering is mainly temporal; non-local spatial similarity is only used to estimate the parameters. This temporally causal method obtains results comparable (in terms of PSNR and SSIM) to methods using several frames per frame denoised, but with a higher temporal consistency. This work has been published in [EMA18].

3.1 Introduction

There are two current trends in video denoising. The first one aims at producing the best video denoising possible whatever the computation required, while the second trend focuses on producing the fastest causal video denoising algorithm with the goal of real-time processing.

In terms of output quality the state of the art is achieved in part by patch-based methods [DFE07, MBFE12, AM18a, EAM17, BLM16, WLPB17]. As we have previously seen in Chapter 1, they exploit the self-similarity of natural images and videos, namely that most patches have several similar patches around them (spatially and temporally). Each patch is denoised using its similar patches, which are searched for in a region around it. The search region generally is a spatio-temporal cube, but more sophisticated search strategies have also been used. Because of the use of such neighborhoods these methods are called *non-local*. While these algorithms perform very well, they often are unpractical: they use a frame's past and future and therefore can only be used off-line. Because of their complexity they are unfit for high resolution video processing.

Fast algorithms rely on much simpler principles which can be implemented efficiently on GPUs or FPGAs. For example [PPR+17] relies on a bilateral filter and a Kalman filter to produce a real-time video denoising algorithm. A recursive version of the non-local means algorithm is proposed in [AH17]. These methods use simple denoising strategies yet generally result in poor denoising quality for high noise levels.

Convolutional networks have been successfully applied to image denoising, and also to other video processing tasks such as deblurring [SDW+17] or video synthesis. Their application to

video denoising has been limited so far. In [CSY16] a recurrent architecture is proposed, but the results are quite below the state of the art. Recently [GMU18a] focused on the related problem of image burst denoising reporting very good results.

In this chapter we present a video denoising method that tries to close the gap between high-quality but slow methods and those efficient but of lower quality. It is much faster than most state-of-the art algorithms (yet not real-time) but in contrast to them it is causal temporally and recursive, i.e. it uses only the current frame and the previous denoised frame. In spite of these strong design constraints it achieves a state-of-the-art performance in image quality.

3.2 Proposed Method

In the following we denote by u a clean video which has been contaminated by an additive Gaussian white noise n of (known) standard deviation σ, i.e. only $v = u + n$ is observed. Even though the additive white Gaussian model seems simplistic, it is enough to also process more realistic Poisson noise. Indeed, Poisson noise can be reduced to nearly white Gaussian by a variance stabilizing transform [ZFSOM07]. The video u, respectively v, is made of f frames indexed between 0 and $f - 1$; the frames are denoted by u_t, respectively v_t, with $t \in [\![0, f - 1]\!]$. We will work with two dimensional square patches of size $s \times s$, a patch will be denoted by a bold lower case letter such as $\boldsymbol{p}$, considered as a vector of dimension s^2.

3.2.1 Framework

Our method works by temporal filtering along patch trajectories using a Kalman filter associated to each trajectory. The Kalman filter requires for its operation to keep track of the state covariance matrix. The state in our case is the clean patch that we want to estimate (arranged as a vector of s^2 components), and the state covariance is a $s^2 \times s^2$ matrix. We use a simple dynamic model for the temporal evolution of the patch trajectories, which requires the estimation of a single parameter at each time step: the *state transition covariance matrix*. It models how a (clean) patch can vary from one frame to the next. Estimating the state transition covariance matrix is in general a difficult problem. We resort to non-locality for obtaining a reliable estimate. To that aim, we group trajectories based on the similarity of the first patch, and assume that these trajectories share the same model. The covariance matrix is obtained from the statistics of the trajectories in the group. This assumption also allows us to reduce the memory consumption of the method, since all the trajectories in the group, because they share the same transition covariance, also share the state covariance. Because there is still a small high frequency residual noise after the NL-Kalman pass, the multiscale DCT denoising algorithm [PMF17] is applied to each frame as a post-processing.

The definition of patch trajectories, and their computation from an optical flow is explained in Section 3.2.2. Patch trajectories can be terminated if an occlusion happens (or more generally due to a registration error). This is explained in section Section 3.2.5. The spatial denoising algorithm is responsible for initializing the groups of trajectories, and is explained in Section 3.2.3. Finally, the key of our contribution, namely the non-local Kalman filters running on groups of trajectories, is explained in Section 3.2.4.

3.2.2 Patch trajectories

A patch trajectory is temporal sequence of $s \times s$ patches $\boldsymbol{q}_0, \boldsymbol{q}_1, \ldots, \boldsymbol{q}_t$ extracted from consecutive frames in the video. The centers of these patches follow a motion trajectory, which is estimated using an optical flow algorithm.

Figure 3.1: Evolution of the quality of the denoising during the warm-up stage. Artifacts due to the spatial denoising quickly disappear after few iterations of the temporal filtering.

Suppose we know the trajectory of a patch until frame $t-1$, $\boldsymbol{q}_0, \boldsymbol{q}_1, \ldots, \boldsymbol{q}_{t-1}$. The next patch in the trajectory at frame t is computed by tracing the optical flow between frames t and $t+1$. We assume that all the pixels in the patch move with the same optical flow as the patch center. This is a crude approximation but it allows for a simpler dynamic model.

We use an optical flow to compute the patch trajectories because it is dense and is not restricted to any particular type of motion (as opposed to e.g. global parametric models). In particular, we used the implementation of [SPMLF13] of the TV-L1 optical flow [ZPB07]. Any other motion model can be used, as long as it provides a dense set of correspondences. Optical flow is less robust to noise than global parametric models. To gain robustness to noise (and speed), we compute it on a downscaled version of the frames, similar to [LYT+14].

3.2.3 Spatial denoising and starting groups trajectories

The spatial denoising is used when no temporal information from the previous frame is available (e.g. in a dis-occluded area). The goal here is not only to denoise these areas, but also initialize the groups of trajectories for the temporal denoising of these areas in the future frames. We use a single step of the NL-Bayes denoising algorithm [LBM13a], which has the benefit of computing groups similar patches and their covariance matrices (needed for the Kalman filters) as part of the denoising.

Given a patch $\boldsymbol{q}$ of the noisy frame v_t, and the corresponding unknown patch $\boldsymbol{p}$ of the clean frame u_t, the following Gaussian linear model of $\boldsymbol{p}$ and $\boldsymbol{q}$ is assumed: $\mathbb{P}(\boldsymbol{p}) = \mathcal{N}(\boldsymbol{\mu}, C)$ and $\mathbb{P}(\boldsymbol{q}|\boldsymbol{p}) = \mathcal{N}(\boldsymbol{p}, \sigma I)$. Once the mean patch $\boldsymbol{\mu}$ and the covariance matrix C have been estimated from the noisy video, the MAP estimate $\hat{\boldsymbol{p}}$ given a noisy patch $\boldsymbol{q}$ is obtained as in [LBM13a]: $\hat{\boldsymbol{p}} = \boldsymbol{\mu} + C(C + \sigma^2 I)^{-1}(\boldsymbol{q} - \boldsymbol{\mu})$.

The parameters of the *a priori* model are learned from the noisy frame. For each noisy patch $\boldsymbol{q}$, N similar patches are selected from the frame. Let $\boldsymbol{q}_i$, $i = 1, \ldots, N$ be the set of patches similar to $\boldsymbol{q}$ (with $\boldsymbol{q}_1 = \boldsymbol{q}$). The estimates for $\boldsymbol{\mu}$ and C are given by $\hat{\boldsymbol{\mu}} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{q}_i$ and $\hat{C} = \frac{1}{N}\sum_{i=1}^{N} \overline{\boldsymbol{q}}_i \overline{\boldsymbol{q}}_i^T - \sigma^2 I$, where $\overline{\boldsymbol{q}}_i = \boldsymbol{q}_i - \hat{\boldsymbol{\mu}}$ are the centered patches.

We call group of trajectories such a set of patches with their C associated. The idea is to follow the evolution of that group of patches by tracking their respective trajectories. We assume a common destiny for all these patches and so we update C jointly for all these patches along the trajectories. This recursive step is presented in the next section.

σ	Method	Bus	Foreman	Pedestrian area	Crowd_run	Touchdown pass	Station2	Average
10	VBM3D*	33.32/.7824	**37.40**/.6681	**40.78**/.6577	35.62/.8017	39.08/.6103	38.92/.7266	37.52/.7078
	VBM4D-np*	**33.39**/.8237	37.39/**.6871**	40.56/**.7463**	**35.69**/**.8457**	**39.60**/.6752	**39.93**/.7746	**37.76**/.7588
	NL-Kalman	33.34/**.8502**	36.16/.6782	38.67/.7420	34.29/.8383	38.82/**.6940**	39.91/**.7916**	36.86/**.7657**
	NL-Kalman (oracle)	*33.87/.8713*	*36.93/.7230*	*39.23/.7592*	*34.64/.8514*	*39.58/.7433*	*40.50/.8059*	*37.46/.7923*
20	VBM3D*	29.57/.6064	34.60/.5763	**36.93**/.5579	**32.22**/.7122	36.09/.4703	35.45/.5689	34.14/.5820
	VBM4D-np*	29.55/.6856	**34.61**/**.6073**	36.75/**.6468**	32.07/.7439	**36.41**/.4795	36.23/.6395	**34.27**/.6338
	NL-Kalman	**29.58**/**.7291**	33.19/.5844	35.61/.6444	30.89/**.7478**	35.91/**.5181**	**36.81**/**.6868**	33.66/**.6518**
	NL-Kalman (oracle)	*30.43/.7752*	*34.18/.6301*	*36.45/.6738*	*31.44/.7746*	*36.99/.6135*	*37.46/.7116*	*34.49/.6965*
30	VBM3D*	**27.59**/.4995	32.77/.5224	34.44/.4869	**30.14**/.6394	34.55/.3906	33.36/.4536	32.14/.4987
	VBM4D-np*	27.53/.5988	**32.91**/**.5612**	**34.45**/**.5745**	29.95/.6704	**34.76**/.3801	34.14/.5420	**32.29**/.5545
	NL-Kalman	27.30/**.6327**	31.27/.5335	33.27/.5680	28.64/**.6708**	33.91/**.4034**	**34.73**/**.5986**	31.52/**.5678**
	NL-Kalman (oracle)	*28.48/.6993*	*32.50/.5802*	*34.43/.6102*	*29.44/.7078*	*35.20/.5186*	*35.46/.6338*	*32.59/.6250*

Table 3.1: Quantitative comparison with VBM3D and VBM4D on grayscale sequences. NL-Kalman (oracle) corresponds to the proposed method using an optical flow computed on the clean sequence. Results with a star were computed using the binary provided by the author. Best results are in bold (results from the oracle are excluded). VBM3D has the lowest running times, followed by NL-Kalman and finally VBM4D. Other state-of-the-art methods like SPTWO [BLM16] and video NL-Bayes [AM18a] can achieve better results, but with a significantly higher running time. See text for details.

3.2.4 Temporal filtering

We propose a recursive temporal filtering. We assume available a set of groups of trajectories estimated from the previous frame (either from the temporal filtering or the spatial initialization) such that the union of all trajectories in the groups covers that frame. To denoise the frame t, we loop on the groups updating the parameters of the Kalman filter and operating it to estimate the clean patches of the trajectories in the group at frame t.

Let $G = \{\hat{\boldsymbol{p}}_{t-1,1}, \dots, \hat{\boldsymbol{p}}_{t-1,N}\}$ be a group of trajectories. For each patch $\hat{\boldsymbol{p}}_{t-1,i}$ we have a noisy observation $\boldsymbol{q}_{t,i}$. The registration test compares $\hat{\boldsymbol{p}}_{t-1,i}$ and $\boldsymbol{q}_{t,i}$ and determines if the registration is correct or not. This is detailed in Section 3.2.5. The trajectories that do not pass this test are terminated and removed from the group. In the following we assume that the N trajectories in the group passed the test. For such patches, we have a prediction, namely the previous denoised patch in the trajectory.

We assume that the evolution of the patch along a trajectory follows a simple linear dynamic Gaussian model, described as follows:

$$\boldsymbol{p}_{t+1,i} = \boldsymbol{p}_{t,i} + \boldsymbol{w}_{t,i} \text{ with } \boldsymbol{w}_{t,i} \sim \mathcal{N}(\boldsymbol{0}, C_t) \tag{3.1}$$

$$\boldsymbol{q}_{t,i} = \boldsymbol{p}_{t,i} + \boldsymbol{n}_{t,i} \text{ with } \boldsymbol{n}_{t,i} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 I). \tag{3.2}$$

Here $\boldsymbol{w}_{t,i} \sim \mathcal{N}(0, C_t)$ is the process noise, modelling the variations of a patch from one frame to the next. It depends on t because we assume that patches can evolve (slowly) in time. The only model parameters that need to be estimated are the state transition covariances C_t associated to the group. It is easy to verify that

$$\mathbb{E}\{(\boldsymbol{q}_{t,i} - \boldsymbol{q}_{t-1,i})(\boldsymbol{q}_{t,i} - \boldsymbol{q}_{t-1,i})^T\} = C_t + 2\sigma^2 I. \tag{3.3}$$

By assuming that the patches in a group are independent realizations of the same dynamic model, we can estimate the expectation as the sample covariance matrix of the *innovation* vectors $\boldsymbol{q}_{t,i} -$

Figure 3.2: Denoising results when the sequence is corrupted by noise of standard deviation 30. From left to right: Denoising using NL-Kalman, NL-Kalman with the oracle optical flow, VBM3D, VBM4D and the original. From top to bottom: Crop from the sequence `crowd_run` and `pedestrian_area`. Overall the proposed method preserves more details as can be seen both in the tree and in the text. Results best viewed zoomed.

$\boldsymbol{q}_{t-1,i}$ of the trajectories in the group. We thus estimate C_t as the following spatio-temporal average:

$$C_t = \beta C_{t-1} + (1-\beta) \cdot \left(\sum_{i=1}^{N} \frac{(\boldsymbol{q}_{t,i} - \boldsymbol{q}_{t-1,i})(\boldsymbol{q}_{t,i} - \boldsymbol{q}_{t-1,i})^T}{N-1} - 2\sigma^2 I \right), \tag{3.4}$$

To increase the temporal stability of the estimate, we combine the non-local average over the group with the previous estimate C_{t-1}, with a forgetting factor $\beta \in [0,1]$. The initial C_0 is set as the covariance of the spatial denoising. We estimate the patches and update their covariance with the Kalman filter equations [Kal60]:

$$\text{Predicted estimate cov. } P'_t = P_{t-1} + C_t \tag{3.5}$$

$$\text{Optimal Kalman gain } K_t = P'_t (P'_t + \sigma^2 I)^{-1} \tag{3.6}$$

$$\text{Update state } \hat{\boldsymbol{p}}_{t,i} = \hat{\boldsymbol{p}}_{t-1,i} + K_t(\boldsymbol{q}_t - \hat{\boldsymbol{p}}_{t-1,i}) \tag{3.7}$$

$$\text{Updated estimate cov. } P_t = P'_t - K_t(P'_t + \sigma^2 I)K_t^T \tag{3.8}$$

Here P_t is the state covariance matrix, it needs to be stored with the group of patches (as the state transition covariance C_t). The dynamic model is initialized using the result of the spatial denoising on the patch group. In particular the initial state covariance matrix is taken as the covariance computed by the NL-Bayes spatial denoising algorithm.

The proposed model can be interpreted as a dynamical version of the video NL-Bayes denoising method [AM18a]. Their similar 3D patches (of size around $7 \times 7 \times 2$) are assumed to be normally distributed. The parameters of these Gaussian distributions are estimated from the set of similar patches, gathered from past and future frames. The estimated covariance matrix represents the modes of variation of the population from the estimated mean patch. Instead our Gaussian dynamical models represent temporal variations along patch trajectories.

3.2.5 Registration quality assessment

We now address the registration quality assessment, to detect occlusions and errors in the optical flow. The quality must be assessed for each patch trajectory individually. The idea is to compare the denoised patch (the prediction in the Kalman filter) to the new noisy observation. Let $\boldsymbol{q}_t$ be the

patch candidate and $\boldsymbol{p}_{t-1}$ the result of the denoising for the corresponding patch at the previous frame. The assumption underlying the registration quality assessment is that $\boldsymbol{q}_t$ is a noisy version of $\boldsymbol{p}_{t-1}$ (Eq. (3.2)). Therefore $\frac{\|\boldsymbol{p}-\boldsymbol{q}\|_2^2}{\sigma^2} \sim \chi^2(d)$. Our goal is to reject patches that are too different, as the dynamic system can only deal with small changes. The optimal test to assess the quality of the matching corresponds to $\frac{\|\boldsymbol{p}-\boldsymbol{q}\|_2^2}{\sigma^2} \geq c$. In practice it is better to also take into account the temporal variation of patches but this will be disregarded here for simplicity.

In order to choose the best possible c, one can use a statistical approach like in [DMM07, VGJMR10, PGvG12, LGvGRM14, MS04, PGvGO13] based on a number of false alarm (NFA). We would like the patch $\boldsymbol{p}$ to be rejected if and only if we are confident enough that it is not a good match and not because of the noise *i.e.* $N\mathbb{P}\left(\frac{\|\boldsymbol{p}-\boldsymbol{q}\|_2^2}{\sigma^2} \geq c\right) \leq \varepsilon$ where ε corresponds to the number of false matches that can be accepted and N to the number of time a patch is added to trajectory. This leads to $c \geq G^{-1}\left(1 - \frac{\varepsilon}{N}\right)$ where G^{-1} the inverse cdf of a chi square distribution of dimension d. A sound value for N is the number of pixels of a frame. Indeed, fixing $\epsilon = 1$ amounts then to admit at most one false matching on average per frame.

3.3 Experiments

We compared the performance of our algorithm against the state-of-the-art denoising algorithms VBM3D and VBM4D. Our causal recursive method requires a couple of frames to warm up. Therefore we remove the first five frames in the computation of the PSNR and SSIM to get a fair comparison. Figure 3.1 shows the quick improvement of the denoising quality in the first three frames used as warm up. Table 3.1 presents denoising results on 1920×1080 sequences from Derf's video database[1] that have been converted to gray by averaging the RGB channels and downscaled by two so as to ensure that they contain little noise. A Gaussian blur was applied before downscaling to avoid aliasing.

Overall the proposed method performs better than VBM3D and VBM4D in SSIM but shows a lower PSNR. The main limitation of NL-Kalman stems from the optical flow. Estimating the optical flow from noisy data is challenging. This causes miss-registration errors which explains the observed drop in PSNR performance. To evaluate this we compare the denoising results obtained with optical flow computed on the noisy data to those obtained from ground truth optical flow. In spite of this, the result of NL-Kalman shows a much higher visual quality, in terms of temporal consistency and in the amount of recovered details. This explains why the difference between the SSIM measure, where NL-Kalman leads, and PSNR, where the average difference is about 0.7db in favor of VBM4D. A comparable result in terms of temporal consistency is attained with SPTWO [BLM16], a more complex method using 10 frames for each frame denoised. SPTWO is also based on the optical flow, but is not recursive. To denoise each frame it computes the optical flow between the target frame and n neighboring frames ($n/2$ past plus $n/2$ future frames). We computed SPTWO with $n = 8$ for $\sigma = 30$ and obtained an average PSNR of 32.45 (SSIM .6114).

Among the algorithms compared, the faster one is VBM3D. Our current implementation in non-optimized C++ code is between two to three times slower than VBM3D. The main reason for this is that the number of groups grows as time evolves, since new groups are created for covering patches in dis-occluded areas. We are currently working on faster versions of the algorithm by limiting the number of groups.

[1] https://media.xiph.org/video/derf/

3.4 Conclusion

We presented a novel patch-based video denoising method and showed that it is competitive with the state of the art. It is based on temporal filtering along trajectories of patches. The temporal filtering is carried out by Kalman filters whose parameters are estimated non-locally. The resulting method is recursive, as to denoise one frame it uses solely information from the previous frame. Results show a performance comparable to more complex patch-based methods that use around ten frames to denoise each single frame. The main limitation of the method is that it relies heavily on the optical flow. Slight errors in registration cause a drop in PSNR, even if visual quality of the result is much superior to the one of related algorithms, both in terms of details and temporal consistency. This work also shows the limits of the PSNR and SSIM as quantitative measures for video quality.

4 Fast patch-based denoising on GPU

We have seen in the previous chapters that denoising is an essential part of any image or video processing pipeline. Unfortunately, due to time processing constraints, many pipelines do not consider the use of modern denoisers. These algorithms have only CPU implementations or suboptimal GPU implementations. In this chapter, we propose a new efficient GPU implementation of NL-means and BM3D, and, to our knowledge, the first GPU implementation of the video denoising algorithm VBM3D. The performance of these implementations enable their use in real-time scenarios. This work has been published in [DE].

4.1 Introduction

As we have seen in Chapters 1, 2 and 3, there are two current trends in video denoising. The first one aims at producing the best video denoising possible whatever the computation required (Chapters 1 and 2), while the second trend focuses on producing the fastest causal video denoising algorithm with possibly the goal of real-time processing (3).

An interesting middle ground that we will present in this chapter is to produce optimized implementations of the well performing denoising methods for optimized hardware such as GPUs or FPGAs. This has often been done for image denoising with optimized implementations of NL-means [MP13, DFBCH11] and BM3D [HK19, WXW18]. However, even though performance is even more crucial for video, little work has been done to optimize video denoising methods. The only exception being for video versions of NL-means [GLA+10].

In this chapter, we propose real-time GPU implementations of an improved patchwise NL-means and BM3D. Both implementations outperform all previous proposed GPU implementations, without diminishing the quality of the results. We also use the same backbone to get the first optimized GPU implementation of VBM3D. The performance of this implementation is sufficient for real-time video filtering. This is achieved by regrouping all the filtering operations (fetching the patch data, the data transpositions, the 1D filterings and the thresholding) into one kernel without requiring an intermediate buffer for the patches being processed. This significantly reduces the use of memory bandwidth. Moreover, all memory accesses are designed so as to reduce bandwidth and benefit from the cache.

The chapter is organized as follows. Section 4.2 presents the studied methods. Section 4.3 presents the specificity of GPU implementations. Section 4.4 presents the efficient GPU implementations for the methods presented in Section 4.2. The efficiency of the proposed implementations are measured in Section 4.5. Finally Section 4.6 concludes.

4.2 Architecture of the implemented Non-local denoising algorithms

Both NL-means and BM3D (and their video extensions) are based on a self-similarity principle. This means that their architectures are relatively similar; built around a patch search step as well as a patch group processing step. We present the different methods in detail in Sections 4.2.1, 4.2.2 and 4.2.3 as well as the different implementation options.

4.2.1 NL-means and its extension to video

Original NL-means. NL-means [BCM05a] is a popular image and video denoising algorithm, due to its simplicity and speed. It introduced the self-similarity hypothesis: for a patch (a small region of the image) of a natural image, many similar patches can be found in its neighborhood. For noisy images, these similar patches will have different noise realizations and therefore their information can be combined to estimate the restored information. The original NL-means works as follows: for each position (x, y) of an image u, consider the corresponding reference patch $P(x, y)$ centered at this position. The value of the estimated image $\hat{u}$ at position (x, y) is computed as a weighted average of all patches in the small local neighborhood $N(x, y)$ centered on (x, y). While the self-similarity hypothesis says that similar patches should be found, not all patches in the local region are similar. The weights are therefore used to reduce the impact of patches that are too different. The original NL-means suggested the weighted average presented in Equation (4.1) where the only parameter h is here to take into account the noise.

$$\hat{u}(x, y) = \frac{1}{C_{x,y}} \sum_{x',y' \in N(x,y)} w_{x,y}(x', y') u(x', y'). \tag{4.1}$$

with

$$w_{x,y}(x', y') = \exp\left(-\frac{\|P(x, y) - P(x', y')\|_2^2}{h^2} \right) \tag{4.2}$$

and

$$C_{x,y} = \sum_{x',y' \in N(x,y)} w_{x,y}(x', y') \tag{4.3}$$

Improved NL-means. In this chapter, an alternative version of NL-means is implemented. The alternative version was preferred since it is more efficient (it requires less computations) while still achieving denoising on par with the original implementation. The improved version follows the same global structure as the NL-means presented in the previous paragraph. The alternative relies on four major differences.

First, we use Equation (4.4) instead of Equation (4.2). This improvement proposed in [BCM11] was shown to reduce a matching noise bias in [FK18].

$$w_{x,y}(x', y') = \exp\left(-\frac{(\|P(x, y) - P(x', y')\|_2^2 - 2\sigma^2)_+}{h^2} \right) \tag{4.4}$$

We decided to limit the number of patches used in Equation (4.1). Indeed instead of using all patches from the local search region $N(x, y)$, only use a subset comprised of the n best nearest neighbors is used. This is possible since most patches in the search contribute very little in Equation (4.1). Very often only a few patches have a distance small enough to contribute. The exception of the flat regions is mentioned later. This choice is principally for computation reasons as it speeds up the method as shown in [MS05] and [CYB06]. However it can also improve the quality of the results, acting as a regularizer as shown in [GO07] and [KBC07].

	Original NL-means	Improved NL-means
Patches used	All patches in the local region	n patches in the local region closest to the reference
Flat patches	Same as other patches	Average all pixels
Weights	$e^{\left(-\frac{\|p-p'\|_2^2}{h^2}\right)}$	$e^{\left(-\frac{\left(\|p-p'\|_2^2-2\sigma^2\right)_+}{h^2}\right)}$
Aggregation	Center pixel of the estimated patch	All pixels of the estimated patch with bilinear weights

Table 4.1: Difference between the original NL-means and the improved NL-means version used for this work.

We also added a "flat patch trick" similar to NL-Bayes [LBM13a]. For that, a simple test on the variance of the n patches found is added. A flat region is detected when the variance of the pixels from these patches is too close to the variance of the noise (see Equation (4.5) with $\beta = 1.05$).

$$Var[\{P_i \mid i \in \{1, \dots, n\}\}] < \beta\sigma^2 \tag{4.5}$$

In that case we simply average the contribution of all pixels to produce the estimated patch (all pixels of the estimated patch have the same value). This means that $n \times p \times p$ contributions are averaged instead of only n contributions where p is the width of the patch. This improves the quality of the denoising in flat regions, especially when the number of patches used is small.

The last improvement is the use of a patchwise NL-means as proposed in [BCM11]. In the patchwise NL-means variant, the entire patch is denoised by averaging and the entire estimated patch is aggregated instead of just its center pixel. However, instead of using the uniform weighting of [BCM11] for the aggregation, we use bilinear weighting. This means that pixels near the center ot the patch will have a greater impact during aggregation. The advantage of the patchwise NL-means variant is that an aggregation step can be introduced to skip some pixels like in BM3D [DFKE07b] and NL-Bayes [LBM13a] and thus limits the amount of computation. The final estimate of these pixels comes from the aggregation of the overlapping results from the other patches. The improved version of NL-means is described in Algorithm 10 (also shown in 4.1) and the differences are summarized in Table 4.1.

This is not the first work to try to improve NL-means running time or improve the denoising performance. Examples of acceleration are [BKC08] which uses a cluster tree for the patch search and [WGY+06] which use FFT and the Summed Squares Image for the filter computation. Examples of denoising performance improvements are [KBC07] which uses a Bayesian model to replace the NL-means weights. Instead of using all the patches in the window, a locally adapted dictionary is extracted locally from the image in a first pass, and from a first denoising result - or oracle - in a second, [DAG10] proposes a local adaptation of the smoothing parameter h using a method based on SURE, [SDA14] improves the output of NL-means, especially on regions with few good neighbors, by minimizing a global cost function involving the NL-means weights, the NL-means output and the TV norm, [JGKL17] replaces the NL-means weights with triangular kernels and a scheme to automatically adapt the smoothing parameter. The authors claim that their proposed kernels can be compared to the one proposed in [BCM11] which we use. However all these works focus on CPU implementations. Indeed their modifications are not adapted for

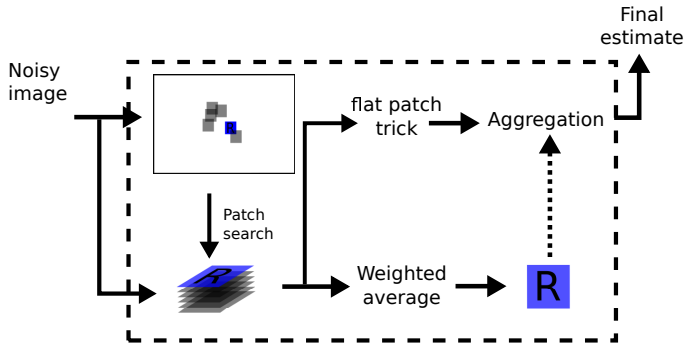


Figure 4.1: Scheme of the NL-means algorithm

GPU contrary to our improved NL-means that is specifically adapted for GPU.

Algorithm 10: Proposed improved NL-means version

Input: Input image u, N the number of similar neighbors, s the patch step

Create an empty image v (using u size)
for *each patch* p *of* u *on a grid of step* s **do**
    Search $p_1, \ldots, p_N$ the N nearest neighbors of p
    if $p_1, \ldots, p_N$ *are all flats (test with* (4.4)*)* **then**
        Average all pixels into a flat patch $\hat{p}$
        Aggregate all pixels of $\hat{p}$ in v
    else
        Compute the aggregation weights w_i for p_i with (4.4)
        $\hat{p} = \sum_{i=1}^{N} w_i p_i$
        Aggregate all pixels of $\hat{p}$ in v
return v

NL-means extension to video. The extension of NL-means to video is simply done by searching patches in a 3D neighborhood as mentioned in [MS05].

4.2.2 BM3D by Dabov *et al.*

The denoising principle of BM3D exploits the redundancy of similar patches. Groups of similar 2D patches are assembled in a 3D stack. A separable 3D orthonormal transform is applied to this stack. The stack is denoised by applying a shrinkage operator to the coefficients in a transformed domain. The transform is selected to reveal sparsity in the transformed domain. Most DCT patch coefficients are for example negligible in natural images. However the transform being a patch isometry, the noise remains spread evenly among all DCT coefficients. The algorithm follows four basic steps:

1. Search for similar patches in the image and group them in 3D stacks,
2. Apply a 3D linear domain transform to the 3D blocks,
3. Shrink the transformed coefficients,
4. Apply the inverse transform,

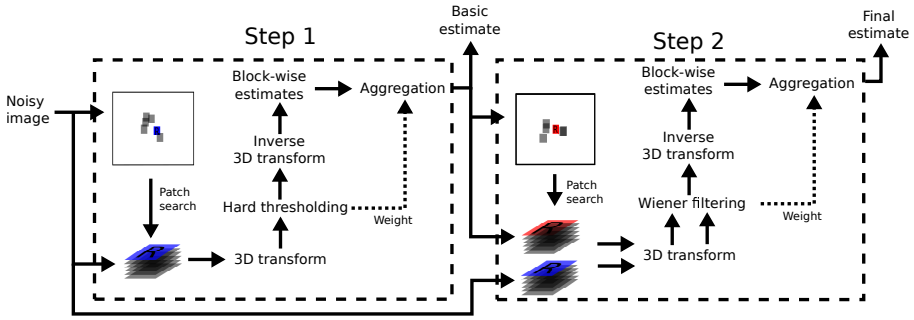


Figure 4.2: Scheme of the BM3D algorithm

σ	Method	Crowd	Park	Pedestrians	Station	Sunflower	Touchdown	Tractor	Average
10	VBM3D (7×7)	35.57	34.66	40.84	38.60	40.18	39.14	37.08	38.01
	VBM3D (8×8)	35.55	34.68	40.88	38.70	40.30	39.16	37.12	38.06
20	VBM3D (7×7)	32.06	31.16	36.88	35.22	36.14	36.13	33.11	34.39
	VBM3D (8×8)	32.06	31.20	36.97	35.33	36.27	36.15	33.19	34.45
40	VBM3D (7×7)	28.40	27.68	32.61	31.84	32.36	33.37	29.45	30.82
	VBM3D (8×8)	28.40	27.69	32.62	31.85	32.36	33.38	29.44	30.82

Table 4.2: Comparison of denoising quality using either 7×7 or 8×8 patches for the Wiener estimation of VBM3D. Both have very similar results and therefore justify our usage of 8×8 patches instead of the suggested 7×7.

5. Aggregate the resulting patches in the image.

The construction of the estimated image is done by aggregation, *i.e.* by combining the results of all the different estimations. This principle is applied twice, once to compute a basic estimate and a second time, using the basic estimate as a guide, for a final estimation. In the second stage, this process is indeed repeated but uses the denoised patches of the first "basic" step to find similar patches. Furthermore, transform thresholding is also replaced by a Wiener denoising using the denoised patch as oracle. The method is synthesized in Figure 4.2. In short, we implement exactly the same method as the one presented in [DFKE07b].

4.2.3 VBM3D by Dabov *et al.*

As we have seen in Chapter 1, VBM3D is a straightforward extension of BM3D to video. The only major difference lies in the patch search. The method was synthesized in Figure 1.1. Since the 7×7 patch size of the second step is difficult to implement efficiently on a GPU, we decided to use 8×8. We checked using the implementation from Chapter 1, that this choice incurs in no significant performance loss. See Table 4.2. This is the only difference compared to the method proposed in [DFE07].

4.3 An introduction to GPU architecture

To understand some of the implementation choices in Section 4.4, some knowledge about GPU computing workloads is required. We shall thus give an overview of the main particularities and challenges of GPU programming. In this section, we define a coherent hardware terminology.

GPUs and OpenCL for highly parallel workloads. Modern GPUs have a highly superior computing power compared to modern CPUs; they are highly multithreaded. There can be 10^5 to 10^6 threads running at the same time. While parallelization of many data processing algorithms is generally straightforward on CPU since CPUs' cores are fairly independent, GPUs need more care to take advantage of their highly parallelized architecture. In OpenCL, the programmer writes *kernels*, which are functions executing on the GPU. All assigned threads execute the same kernel code but start with a different work-item index. This leads each kernel to do different portions of the work that needs to be parallelized. Alongside kernels, an OpenCL program also needs CPU code to manage the GPU memory and define when and with which arguments these kernels should be executed.

The programmer has some control on how indices are assigned to GPU threads. First, threads are assigned linearly. This enables optimizations so to have consecutive threads access consecutive memory regions. Second, the programmer can define a *workgroup* size (called *local size*). The global size is divided into regions of size defined by the local size. Each region is assigned to a workgroup. The main interest of a workgroup is that its threads have access to a common *local shared memory* only visible to them. This memory enables fast communications between threads. It can be used to store common computations or to cache memory accesses.

A single instruction multiple threads (SIMT) model When assigning indices to threads, the hardware groups consecutive indices of a same workgroup into a *warp*. Each thread inside a warp will execute in *lockstep*. Thus every thread will execute the same instruction at the same time. For example their memory accesses will be simultaneous. This knowledge can be used to reduce the cost of memory accesses. However this model also implies that if several threads need to take different code paths – this phenomenon is called *thread divergence* – all code paths taken need to be executed by the warp, disabling the threads that should not take this code path.

Memory accesses with GPUs Memory accesses are affected by both *latency* and *bandwidth*. Latency is the time taken for an operation to complete. GPUs have two ways of hiding latency. First the code instructions can be organized to have non-dependent computations inserted between a memory access and the code requiring its result. In this case the hardware only needs to wait if the memory access has not finished when the code requiring its results needs to be executed. Second several warps are scheduled on the same computing resources. When a warp has to wait, other warps can be executed.

The bandwidth is the maximum amount of data the hardware can read or write from the memory per second. To maximize the usage of available bandwidth, optimized memory access patterns are required. Any access to read memory at a given location actually loads a batch of consecutive elements (called *cache line*).

Memory accesses are cached to improve latency and bandwidth. Reading the same data again while it is still in cache enables faster access. However, GPU caches are very small relative to the number of threads executing. To compensate for this, GPU threads have access to a significant number of registers, which are local storage only accessible by a given thread.

More details on GPUs can be found in technical documents such as [Jun15, AMD15, NVI09].

4.4 Implementation details

4.4.1 Patch search with a large, 2D or 3D, fixed window

In this section we shall describe how to implement an efficient patch search in a predefined 2D (or 3D in the case of video) search window. The problem can be described as such: Given an

image subgrid with a pixel spacing of $step$ pixels, we want to compute the n best neighbors for each position (the distance between patches used can be L_1 or L_2 for example). The patch is of size $p \times p$ and the $w \times w$ search window is centered on the reference patch. In the case of a video, a temporal depth is added to the window. Typical values for the parameters are $p = 8$, $w = 21$, n in $\{8, 16, 32\}$ and $step$ in $\{3, 4\}$.

Design choices

Implementing this algorithm raises several challenges. Memory accesses must be optimized to make optimal use of the cache. Indeed some computation is common for neighboring patches when $p > step$ and therefore can be shared. Writing in memory all the computed distances would be heavy and thus only a table of the best n patches must be maintained during patch comparison. Only the positions (and if needed the distances) of the n best patches are written at the end.

For the best efficiency, tables of best patch positions and their weights must be stored in GPU registers. This leads to solutions where one thread will manage all the best neighbors for a given location. This also discourages solutions where a given thread would compute, for a given window offset, the distances between several reference patches and their compared patches. Or solutions where a given thread would handle all distances for a compared patch rather than a reference patch. Thus in the following a given thread will always track distances for a fixed reference position.

When a given thread computes the distances between a reference patch and patches in a window, a lot of data is shared between the computations: the reference patch data is used for all computations and most of the data is in common between a compared patch and its neighbor. The distances must however be recomputed from scratch as different pixels are compared. Because of the restricted register space, storing in registers the equivalent of two $p \times p$ patches per thread may not be possible or be desirable. This encourages spreading distance computations for a given reference patch among several threads. These threads would store a part of the reference patch as well as a part of compared patches and reuse them across computation.

Efficient nearest neighbor table Since writing all patch distances in memory would not only require a lot of memory space, but also bandwidth, we want to only store the final best n neighbors. This implies at any point of the algorithm to remember the best n neighbors seen so far (their positions and their distances), and forget the other positions and distances.

On many hardware, accessing an array with a dynamic index (an index unknown at compilation time) requires storing the array in memory rather than in registers. Even if the array ends up staying in cache, the accesses would suffer from latency and bandwidth limitations. Thus to prevent suboptimal performance, the table accesses must always be done with a fixed index. When accessing the array with an index depending on a loop counter, for example, the loop must be unrolled (compiler hints enable that operation). Unrolling the loop means the generated code does not contain the loop jump. Instead the loop content is repeated for each loop index value, thus the array index is known during compilation. Small arrays with only fixed indices accesses can have their content assigned to registers instead of memory.

We propose two alternative algorithms to keep the n best distances and positions: Algorithms 11 and 12. Keeping an ordered table (Algorithm 11) has several advantages over an unordered table (Algorithm 12): The maximum distance is known directly and does not need to be recomputed, and thread divergence is lower. Indeed in Algorithm 12, threads are more likely to insert their elements at different indices than in Algorithm 11.

We found Algorithm 11 to be slightly faster indeed in our tests on a NVIDIA GPU. However, in our tests on INTEL and AMD GPUs, the performance was significantly lower due to the com-

Algorithm 11: Keeping an ordered table of distances and positions

Input: New position pos and distance $dist$, two tables Positions and Distances of length n each.

```
if dist < Distances[n-1] then
    for i from n-1 to 1 do
        insert ← Distances[i-1] ≤ dist
        Positions[i] ← pos if insert else Positions[i-1]
        Distances[i] ← dist if insert else Distances[i-1]
        quit function if insert
    Positions[0] ← pos
    Distances[0] ← dist
```

Algorithm 12: Keeping an unordered table of distances and positions

Input: New position pos and distance $dist$, two tables Positions and Distances of length n each.

```
max_distance ← max(Distances[0], ..., Distances[n-1])
if dist < max_distance then
    for i from 0 to n-1 do
        if Distances[i] = max_distance then
            Positions[i] ← pos
            Distances[i] ← dist
            quit function
```

piler reorganizing the iterations in an unoptimized way for this algorithm, while not having issues with Algorithm 12.

Naïve algorithm

In this naïve algorithm, each thread is assigned a different reference patch for which to compute all the distances and maintain the best n neighbors. No particular effort is spent to encourage the compiler to use registers as a cache to reduce the number of memory accesses.

Threads of the same workgroup are assigned to reference patches on the same row. Thus all memory accesses are done on the same image rows, to minimize the number of cache lines per memory accesses in a warp, and more generally improve the cache usage in our scenario. The efficient nearest neighbor table presented above is used.

Convolution algorithm

This algorithm conceptually implements distance computations as separable convolutions while avoiding storing distances. Threads of a workgroup get assigned consecutive positions on a row. For each window offset, each thread will compute the distance between the p pixels of the column at its reference position and at the compared position. This partial result is written in local shared memory. Each thread then reads the results of neighboring threads in order to determine the distance for the whole current $p \times p$ patch. Only threads with positions on the image subgrid maintain the best n neighbors. For $step = 1$, this algorithm is the best performing among the proposed algorithms, due to its very efficient work-sharing and memory pattern. However for higher $step$s, some computation resources are wasted as a lot of threads do not maintain any nearest neighbor table, an operation with non-negligible cost (due to the phenomenon described

in Section 4.3). Note that no threads are assigned to rows which do not intersect with the image subgrid of reference patches.

As p is small enough, the reference pixels can be stored in registers and not reloaded. As for the pixels of the compared patches, on systems with fast local shared memory, memory accesses can be sped up by storing once (and reusing multiple times) the data required for all the comparisons for a given row of the search window.

Square algorithm

This algorithm is optimized for the case $p = 8$ and $step = 4$. In that specific case, the distances between a reference patch and its compared patches can be split into four distances of 4×4 smaller patches, and these sub-distances are exactly shared among four reference patches. Thus this algorithm proposes to divide the image subgrid into 16×16 squares. Each square is assigned to a workgroup (thus workgroups of 256 threads). Each thread tracks the distances for a given reference patch, but only needs to compute the distances for the top 4×4 region of its reference patch. Each thread writes its intermediate result to local shared memory. They can then retrieve the distance for the entire 8×8 patch by reading the results from three other threads. Naturally, threads at the bottom and right borders of the square workgroup cannot compute the full distances, as no thread computed the required remaining distances outside the square. Thus the workgroup computes the neighbors for a 15×15 region of the subgrid. Some overlapping is required to cover the whole image.

This work subdivision reduces efficiently memory accesses (the reference 4×4 patch can be stored in registers, and the memory accessed for the compared patch can be reused for the next patch and kept in registers). Moreover, threads can use memory commands to load four consecutive pixels per call, which reduces the memory instruction count.

Column computations algorithm

This algorithm is an extension of the convolution algorithm, adapting ideas from the square algorithm. It is less efficient than the square algorithm due to p not being as neatly divided by $step$, but is more efficient than the convolution algorithm when $step > 1$. Similarly to the convolution algorithm, a group of threads is assigned on a row to compute distances between a reference column and a compared column each. However, contrarily to the convolution algorithm, each thread computes not one, but $k + 1$ distances. The columns are composed of $p + k * step$ pixels, and the computed distances are the $k + 1$ partial distances for the intersecting reference patches on the subgrid. The partial results are written in local shared memory and some threads (possibly the same ones) access the results to track the distances for a given reference patch. While for the convolution algorithm, the number of reference patches tracked per thread was low (on average $1/step$), in this algorithm k can be controlled to have almost one patch tracked per thread.

Optionally, the partial distance computations and the neighbor tracking can be done on different threads (from separate warps preferably to not waste computational resources), and thus the total register need can be reduced per thread. Indeed if threads do either the partial distance computations or the best distances tracking, the registers used to keep the best weights and distances can be the same as the ones used to store the column reference pixels for example. While for some algorithms and hardware, it is advised to use a low number of registers to have better memory latency reduction, in our case performance was not affected. We believe latency was not an issue for our algorithm.

4.4.2 VBM3D's patch search

In this section we shall describe how to implement an efficient video patch search on the model of VBM3D's patch search. This patch search is similar to the one of Section 4.4.1. However this time the search is performed in multiple frames of the video. First, the q most similar patches are found in each frame. Then, the n best among these neighbors are kept for the rest of the processing. The reference frame (frame containing the reference patch) has a search window of size $w_1 \times w_1$ centered on the reference patch while the other frames have search regions that are the union of q search windows of size $w_2 \times w_2$ centered on the best neighbors of the previous frame. Each time the reference patch is of size $p \times p$ and is sampled on a subgrid of pixel spacing of $step$ pixels. Default VBM3D parameters are $p = 8$, $w_1 = 7$, $w_2 = 5$, $q = 2$, $n = 8$, $step = 6$ for the first pass and $step = 4$ for the second.

Design choices

The challenge of this patch search compared to the one of Section 4.4.1 is that, except for the reference frame, the search windows are shifted by an offset that depends on the patch. While threads treating patches on the same row of the subgrid would access the same rows at the same time for Section 4.4.1, in this section different rows would be accessed because of the patch-dependent shifts. Moreover a $step$ of 6 means there is quite a gap between the areas accessed by the threads, thus more cache lines would be accessed if using such a partition.

To keep efficient memory access patterns, several threads can be assigned to the same reference patch. Threads can either handle comparisons for different parts of the search windows or the reference patch can be separated into subpatches where each thread is assigned the comparisons of one subpatch. In both cases, the partial results computed from each thread must be communicated to a main thread that will combine them.

Naive algorithm

For this naive algorithm, each reference patch is assigned one thread that computes all distances and determines the n best neighbors with the required temporal constraints (a maximum of q neighbors per frame). Except for the differences in handling the search region, it is essentially similar to the naive algorithm of Section 4.4.1.

Subdividing the test window

For this algorithm, t threads are assigned per reference patch. For example t can be set to w_1 or w_2. The search windows are divided into columns, and each thread searches for the q nearest neighbors in its assigned column(s). One thread combines the results of the other threads using local shared memory. This work partition optimizes the memory access pattern. Indeed all t threads will access neighboring data.

However, as said in Section 4.4.1, due to the restricted register space, storing the equivalent of two $p \times p$ patches in registers may not be possible or desirable. Thanks to the subdivision in columns, the number of distances computed per thread is small enough so that distances can be computed in several steps by storing partial results in registers. In order to do that, the patches are divided into l blocks (composed of columns of the patch). We compute all the distances for a given block, then the next, until the whole distances have been computed. With this method, the partial patch data is small enough to be kept in registers during the computation of the partial distances for a given block. This optimization reduces the number of memory operations significantly.

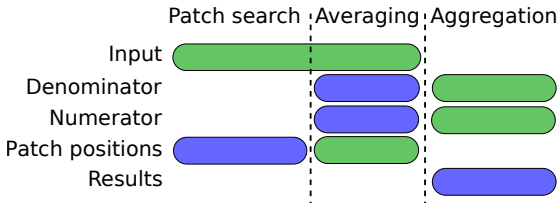


Figure 4.3: Memory layout of NL-means. Buffers in green are read-only and buffers in blue are read-write. Note that there is no intermediate buffer for the processing of the patches.

Subdividing the tested patches

For this algorithm, p threads are assigned per reference patch. Each thread computes all partial distances for one column of the patch. The partial distances are added to get the actual distance (*reduce-add*). Since p is small enough, each thread can keep the reference data in registers and thus avoid reloading the full compared data every time a different position is tested. Moreover the memory accesses are optimized as each thread loads consecutive data.

Compared to Section 4.4.2, this algorithm requires much more communication between threads. However, some hardware support specific fast instructions to share data between threads, which can be used to implement an efficient reduce-add operation. For hardware with such support, this variant can be faster than the previous algorithm.

4.4.3 NL-means' averaging and aggregation

Design choices

Once the best neighbors and their distances are computed, patch distances need to be converted to NL-means weights. We chose to do this computation before saving the results of the patch search, thus saving directly NL-means weights. While it is possible to do both averaging and aggregation at the same time (using a scheme inspired from Section 7), we did not obtain good performance. Thus we separated the averaging and the aggregation steps. The memory layout of the proposed improved NL-means is presented in Figure 4.3.

Naive averaging

In the naive algorithm, a thread computes the averages for a single patch. In order to keep register usage low, all computations are divided per row, moving to the next row only when the results of the previous row has been saved for aggregation.

Optimized averaging

GPUs have specific instructions to access several consecutive data elements per threads. However depending on the data type and the patch width p, it is not possible to read or save a whole patch row in a single memory instruction when using a single thread. Keeping that in mind, in this optimized variant several threads are used to reduce even further the number of memory commands required. Instead of assigning one thread per patch, p threads can be assigned per patch. Each thread handles the averaging for a column of the patch. Reading or writing a row is done in one memory command, thus guaranteeing optimal memory access patterns. The resulting kernel is summarized on Algorithm 13.

Algorithm 13: Proposed NL-means filtering kernel

Input: Input image u, the output accumulator's numerator num and denominator den, the positions of the nearest neighbors $positions$ and the corresponding weights w, the reference patch position and the column j to process

```
1  mean ← 0
2  var ← 0
3  for i from 0 to p-1 do
4      for k from 0 to n-1 do
5          data ← u[k-th neighbor row i col j]    // for patches of width p, p
            consecutive threads access the same patch at
            consecutive columns
6          mean ← mean + data
7          var ← var + data * data
8  mean ← reduce_add(mean) / (p*p*n)
9  var ← reduce_add(var) / (p*p*n)        // The reduce operation is only
     among the p threads treating a same patch
10 var ← var - mean * mean
11 flatpatch ← var <= βσ^2
12 for i from 0 to p-1 do
13     if flatpatch then
14         denoised_pixel ← mean
15     else
16         denoised_pixel ← 0
17         for k from 0 to n-1 do
18             denoised_pixel ← denoised_pixel + w[k-th neighbor] * u[k-th neighbor row i
                col j]
19     Accumulate denoised_pixel on num and den with bilinear weighting
```

Aggregation using atomics

For NL-means, only the results of the denoised version of the reference patch are aggregated (contrarily to BM3D and VBM3D where all similar patches are denoised and aggregated). Thus, the number of patches covering a given pixel aggregated together is known in advance. In this case, it is not required to keep count of the number of covering patches for each pixel and their weights. Each result can be added with atomics on a single buffer, which will then be normalized correctly for each pixel. Atomics are necessary because several threads from different warps can write at the same positions. However, the normalization code is required to handle explicitly all patch sizes and $step$. Thus our code does not implement this optimization, and simply uses an accumulation buffer for the sum of the weighted denoised patches, and a second accumulation buffer for the sum of the weights. Then a simple shader divides the former by the later.

NL-means weights being floats, the averaging results are floats. Depending on the data dynamic and the required precision, int or long atomics can be used to aggregate data thresholded up a given precision (for example 1.000123 gets written as the integer 10001). Float atomics can also be used, but unfortunately to date most hardware do not support float atomics and hardware which do don't support them in OpenCL. Emulating float atomics can be done with int atomics, but with a significantly higher cost. Indeed, a float add needs to be implemented as several int atomic operations: 'reading the data', 'writing the data plus our result if the data is still set to the one we read previously' (this operation returns the value of the data before the operation). This needs to be repeated if the return value of this last operation is not what we expected (it means other threads updated the value after you accessed it). This atomic float add emulation is

Algorithm 14: Emulated atomic float add

Input: Write address p and content to add a

```
1 current ← p[0]                                          // Reads initial value
2 repeat
3 |   expected_current ← current
4 |   next ← expected_current + a                          // Compute float add
5 |   current ← atomic_cmpxchg(p, expected_current, next)   // Try to update
6 until as_uint(current) = as_uint(expected_current)       // Repeat if value
    changed before updating
```

presented on Algorithm 14.

Aggregation without atomics

As said in Section 19, for fixed p and $step$, only one accumulation buffer is enough for the aggregation since the patch aggregation weights are known in advance. If $step$ is large enough, for example $p = 8$ and $step = 4$, it can be advantageous to not even use an accumulation buffer and thus avoid atomics completely. Instead of having a single buffer containing the aggregated value, we write the different values to aggregate per pixel to several different buffers (four buffers in the example above). These buffers can then be read, summed and normalized all a once to get the final output.

Algorithm 15: Proposed BM3D filtering kernel (for the first pass, when the maximum number of neighbors is 8, the 2D transform is the DCT and the 1D transform is the Hadamard transform)

Input: Input image u, the output accumulator's numerator num and denominator den, the positions of the nearest neighbors $positions$, the reference patch position to process for this kernel call

1 **Retrieve** the patch stack from $positions$ `// Each of the 64 threads starts with a table` T `of 8 elements containing one row of the stack`
2 **Apply** the 1D DCT on T
3 **Apply** Algorithm 18 on T
4 **Apply** the 1D DCT on T
5 **Apply** Algorithm 19 on T
6 **Apply** Hadamard on T
7 **Apply** BM3D's hard thresholding on T
8 **Apply** Hadamard on T `// Hadamard is its own inverse`
9 **Apply** the inverse of Algorithm 19 on T
10 **Apply** the 1D DCT on T `// The normalized DCT is its own inverse`
11 **Apply** Algorithm 18 on T `// Algorithm 18 is its own inverse`
12 **Apply** the 1D DCT on T
13 **Accumulate** the results on num and den with BM3D's weighting scheme

4.4.4 BM3D and VBM3D's filtering and aggregation

Design choices

Both BM3D's first and second pass as well as VBM3D's first pass use 8×8 patches. While VBM3D uses 7×7 patches for its second pass. We believe implementing VBM3D's second pass using 8×8 patches does not change the algorithm performance (see Section 4.2.3). For this reason, we will focus on using 8×8 patches only. The number of considered patches per location is always a power of two, 1, 2, 4 or 8 (VBM3D and BM3D), 16 or 32 (BM3D).

One way of solving the filtering is first to read the nearest neighbors for each position, form the associated 3D stack of patches, and write it to memory. The filtering can then be implemented as successive passes on the buffer containing all the stacks. However, the bandwidth needed to write and process the 3D stacks would lead to a suboptimal algorithm. Instead we propose to use the high number of registers available on a GPU to save bandwidth. All considered hardware have workgroups of 64 threads and at least 128 registers available per thread. Thus it is possible to store the $8 \times 8 \times 32$ elements of a 3D stack of patches in the registers of the threads of a given workgroup.

Our proposal is to assign a workgroup of 64 threads for each position to process. The 3D filtering can be implemented as three separate 1D filtering operations (one for each dimension), followed by a shrinking operation (thresholding or Wiener) and finally the three separate inverse 1D filtering operations. At all times, one thread only has direct access to a subset of the 3D stack, such as a few rows. To perform the 3D filtering, each thread performs a 1D filtering operation, and then exchanges its data with the other threads so to have a new dimension of the data (for example columns instead of rows). We call that operation *transposition*. A transposition, for example, moves from a state where each thread contains one (or a few) patch column(s) to a state where each thread contains one (or a few) patch row(s). Implementing efficient 1D filtering or shrinking operations is not hard, thus in the following we focus on the transposition operations.

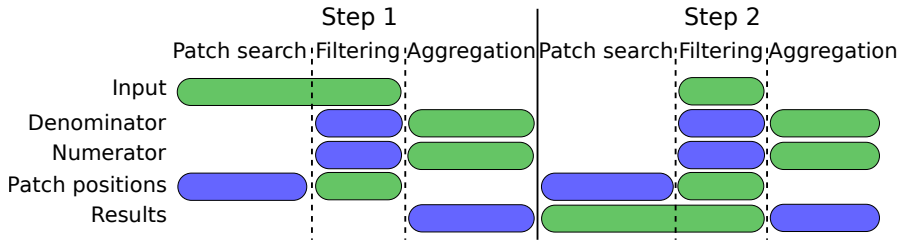


Figure 4.4: Memory layout of BM3D and VBM3D. Buffers in green are read-only and buffers in blue are read-write. Note that there is no intermediate buffer for the processing of the patch stack.

The overall pseudo-code of BM3D's filtering kernel is presented in Algorithm 15 for the specific case of the first pass with a maximum number of 8 neighbors. For 16 or 32 neighbors, each thread reads not one, but two or four rows, respectively, and the operations are repeated for each. The memory layout of the proposed design choices is presented in Figure 4.4.

Efficient transpose operations

We describe here the case of eight patches (of width $p = 8$) and will then extend to the other number of patches.

Initially, each of the 64 threads contains a column of one of the patches. Thus each thread contains eight elements and can apply one of the 1D transforms.

swapping columns and rows The first step is to swap the data so that each thread gets one row of one of the patches instead of a column. To proceed with the swap, each thread writes the eight pixel values it has in local shared memory, which is a fast memory reserved for the 64 threads. Then each thread can read from the local shared memory its target eight pixel values. Doing that efficiently is not as simple as it sounds. Extra care must be taken when reading and writing data so to achieve good performance.

Let us first formalize the transpose operation. It starts with values stored in a table T of size $[8][8][8]$ representing the 3D patch stack. T is stored in the threads, each thread containing the section $[z_i][.][y_i]$ (corresponding to the column y_i of patch z_i) where i is the thread index (from 0 to 63), and z_i, y_i thread coordinates, with $y_i = i \bmod 8$ and $z_i = i/8$. The goal is to have the data reorganized as in a second table T'' such as $T''[z_i][y_i][.] = T[z_i][.][y_i]$. In practice the memory of T is reused for T''.

The most "naive" way of performing the swap is to simply write T in local shared memory in an arbitrary order, and then read the indices in the right order. Examples of such "naive" algorithms are described in Algorithms 16 and 17. Both algorithms are equivalent.

Algorithm 16: Swapping columns and rows (Naive - variant 1)

Input: z_i, y_i thread indices, T thread-specific table of length 8, T' table in local shared memory

for k **from** *0* **to** *7* **do**
  $T'[64k + 8z_i + y_i] \leftarrow T[k]$
barrier() `// Wait for all threads to finish writing`
for k **from** *0* **to** *7* **do**
  $T[k] \leftarrow T'[64y_i + 8z_i + k]$

Algorithm 17: Swapping columns and rows (Naive - variant 2)

Input: z_i, y_i thread indices, T thread-specific table of length 8, T' table in local shared memory

```
1 for k from 0 to 7 do
2 |  T'[64y_i + 8z_i + k] ← T[k]
3 barrier()                  // Wait for all threads to finish writing
4 for k from 0 to 7 do
5 |  T[k] ← T'[64k + 8z_i + y_i]
```

We shall analyze more in depth Algorithm 17. In this naive version, the data is written into an intermediate 3D table T' of size 512 in local shared memory. The table T' is organized as a buffer of size $[8][8][8]$. Each thread (z_i, x_i) writes its data so as $T'[y_i][z_i][.] = T[z_i][.][y_i]$. It just remains to read T' in the right order to obtain T'', $T''[z_i][y_i][.] = T'[.][z_i][y_i]$. Elements are written and read one by one in each thread (loop on k in Algorithm 17), thus resulting into eight write instructions and eight read instructions for each thread (ignoring the instruction merging mentioned later). While at read time, the threads load 64 consecutive elements and thus guaranteeing no bank conflicts, at writing time the proposed pattern triggers bank conflicts Indeed threads 0, 4, 8, etc, write in their first call at cases 0, 32, 64, which map to the same memory bank (all current hardware have either 16 or 32 banks). The same problem happens with the other similar groups of threads. The bank conflicts cause the accesses to the same banks to be serialized, thus being executed as several separate calls. The write performance will thus be suboptimal. The solution to improve performance is to 'shift' the locations of the data written by the threads. The corrected algorithm is identical except T' has padding. It is of size $[8][64 + x]$, but is interpreted as a table of size $[8][8][8]$ as before, but with an offset of x times the index of the first dimension.

This performance trick is well documented and is described for example in [NVI09], which suggests $x = 1$ in the case of a matrix transpose. However, it does not give the best performance on all hardware: Indeed optimal performance is achieved by maximizing the use of banks per calls. However, some hardware can write or read several consecutive elements per thread in one call. The tested INTEL hardware, which can have a warp size of eight, can support reading or writing four elements per thread in one call. Reading or writing only one element results in suboptimal performance as the banks are underutilized. The tested AMD hardware, which has a warp size of 64, executes local shared memory commands in four parts, first the first 16 threads, etc, until the last 16 threads, and ideally each of these subcalls should use all the banks (they have 32). Like for the INTEL GPU, the tested AMD GPU can write or read several consecutive elements per thread in one call (two per call). The tested NVIDIA hardware has warps of 32 threads, has 32 banks, and does not process them in two subcalls. Note that we assume that a bank covers elements of four bytes, which is the size of a float (The NVIDIA hardware can configure it to eight bytes).

Based on this variety of behaviors, it is difficult to design read or write patterns that are optimal on all hardware as it depends on the warp size, the number of banks and how they are accessed. However, assuming the number of banks is a divisor of 64 - our workgroup size -, we noticed that having each thread write (or read) the maximum number of consecutive elements it can write (or read) in one instruction is always an optimal pattern. The same can be noticed when having threads write (or read) from consecutive addresses modulo 64 (thus to consecutive banks). We will use these access patterns in our algorithms for best performance and compatibility (we expect these patterns will still be optimal for future generations).

This leads us to Algorithm 18. The best x corresponds to the maximum number of consecutive elements the hardware can read or write per thread per access. Thus, for the tested hardware, $x =$

Algorithm 18: Swapping columns and rows

Input: z_i, y_i thread indices, T table of 8 registers for the thread, T' table in local shared memory

for k **from** *0* **to** *7* **do**
  $T'[(64+x)y_i + 8z_i + k] \leftarrow T[k]$
barrier() `// Wait for all threads to finish writing`
for k **from** *0* **to** *7* **do**
  $T[k] \leftarrow T'[(64+x)k + 8z_i + y_i]$

4 should be best for INTEL, $x = 2$ for AMD, and $x = 1$ for NVIDIA. We verified experimentally that indeed these parameters proved to be the best performing for each hardware. We also checked the generated assembly code to confirm that the compiler combines the writes of consecutive data into write packets of x elements. We expect the best value of x to change for future hardware generations and should be set accordingly. When memory organization and the memory command packing abilities are not known for a specific hardware, the best x can be determined empirically.

The proposed writing order covers all the banks per groups of threads in a warp and uses all the banks. It is therefore the best possible pattern. For example, on the INTEL hardware with $x = 4$, the first write of thread 0 writes to banks 0, 1, 2, 3, thread 1 writes to banks 4, 5, 6, 7, etc. This corresponds indeed to the optimal pattern we suggested.

Swapping rows and channels Thanks to the transpose presented previously, the first two 1D transforms can be applied on both the rows and the columns. The last 1D transform must be applied on the third dimension of the 3D patch stack. This means that each thread must contain the third dimension of the 3D stack for a given patch pixel. The only constraint for this transposition is for the position $(0, 0)$ of each transformed patch. The rest of the elements does not require any specific ordering as long as the inverse transposition puts back the data in the correct thread. In our case we will require that the position $(0, 0)$ of each transformed patch is put in the thread 0. Furthermore we will assume x is either 1, 2, 4 or 8.

To reformulate the problem, we start with data as a table T of size $[8][8][8]$ stored in the threads. Each thread starts with the section $[z_i][y_i][.]$ where i is the thread index, and z_i, y_i thread coordinates. We want the data to be stored as $[.][\mu_i][\nu_i]$, with μ_i, ν_i thread coordinates. We will use local shared memory as intermediate table T' of size $[8][8/x + 64][8][x]$, and assign

$$T'[z_i][j/x + z_i * 8][y_i][j \bmod x] = T[z_i][y_i][j].$$

This writing pattern verifies the conditions set in the previous paragraph. It avoids bank conflicts and enables full use of the ability of the hardware to write x elements in one call. At reading time, we assign $T[j][z_i][y_i] = T'[j][a_i + j * 8][b_i][c_i]$, where $a_i * 8 * x + b_i * x + c_i = z_i * 8 + y_i$. As among threads, y_i increases first, then z_i , the pattern also prevents bank conflicts (it verifies again the conditions set previously). Contrary to when writing T', the read commands cannot be combined to read more than one value per call.

One problem with this proposed algorithm is the size of T' which can be huge: It is of length $512(1 + x)$. However T' can be fit in a much smaller table of size $[8][8/x][8][x]$, that is of length 512, while maintaining an optimal access pattern. To do so, modulus will be used in the writing and read address computations.

Algorithm 19 presents the entire algorithm (with T' a 1D table of length 512). It is easier to visualize on Algorithm 19 that the algorithm is correct and that it indeed gives the pattern we described in the previous paragraph: Since $(8z_i + y_i)$ indexes threads from 0 to 63, the loops of

Algorithm 19: Swapping rows and channels

Input: z_i, y_i thread indices, T table of 8 registers for the thread, T' table in local shared memory

for h **from** *0* **to** *(8/x - 1)* **do**
  for *j* **from** *0* **to** *x-1* `// The x memory operations are merged`
  do
    $T'[64z_i + (8z_ix + 8hx + y_ix + j) \bmod 64] \leftarrow T[hx + j]$
barrier() `// Wait for all threads to finish writing`
for k **from** *0* **to** *7* **do**
  $T[k] \leftarrow T'[64k + (8kx + 8z_i + y_i) \bmod 64]$

Algorithm 19 access indeed consecutive addresses modulo 64 for a fixed h and a fixed k. Thus optimal performance is guaranteed.

Generalizing to 1, 2, 4, 16 or 32 patches The swapping operations presented previously assumed we have to process eight patches. In order to generalize to more patches, we simply need to store several columns of data initially per thread, and call the transposition functions presented previously several times. When having fewer patches, some operations could be avoided. However, we simply called the function for eight patches while setting the missing patches to 0 and therefore just ignoring them in the 1D transforms.

On the use of atomics for the aggregation

Due to the non-uniform weighting, the weighted pixel values being aggregated have a wide range and, contrarily to NL-means, many elements are aggregated per pixel for BM3D and VBM3D. We found that even with 8-bit RGB images, 32 bits integers could be insufficient to store the weighted sum in some corner cases. Long atomics (64-bits integers) are recommended. When not supported, the emulation of float atomics discussed in the Section 19 should be used. Integer atomics could be used if reducing the range of the weights used for the aggregation, though the PSNR could be affected.

4.4.5 Conclusion

In this section, we have explained in detail different optimized implementations (as well as their naive versions) of the pieces of code required for NL-means, BM3D and VBM3D. In our implementation, which we compare to other implementations in Section 4.5, we have selected the best optimized codes among the ones described, depending on the parameters. For the patch search of NL-means and BM3D, we use the fast variant described in Section 7 when possible ($p = 8$, $step = 4$), else use the variant of Section 7, except for $step = 1$, in which case the variant of Section 7 is fastest. The naive variant is only used for border handling when needed. For NL-means, we use the optimized averaging, rather than the naive one. We do aggregation using atomics for all methods. Last, VBM3D's patch search uses Section 4.4.2 for INTEL GPUs, else the algorithm of Section 4.4.2.

4.5 Benchmarks

In this section we will compare the performance, both in terms of running time and PSNR, of our implementations with several reference implementations. To remove the effects of differences

Acc.	Description	Bandwidth	Float ops
0	Intel Xeon W-2145 CPU (18.1.0.0920)	63 GB/s	1.8 T/s
1	INTEL i7-6600U GPU (NEO 18.21.10858)	23 GB/s	380 G/s
2	AMD Radeon RX 480 (2766.4)	209 GB/s	5.7 T/s
3	NVIDIA TITAN V (390.129)	611 GB/s	13.7 T/s

Table 4.3: Compared configurations: The first accelerator is a 16-core CPU, while the other accelerators are GPUs with different levels of performance. The bandwidth and number of operations per seconds were obtained with Clpeak, which estimates the actual peak figures which can be obtained with OpenCL.

Method	BSD68	Kodak	IPOL
[BCM11]	28.81	29.93	31.89
OpenCV	28.27	29.33	31.53
OpenCV GPU	28.24	29.31	31.55
Ours P5,S1	28.92	30.13	32.60
Ours	28.51	29.76	32.12

Table 4.4: Comparison of NL-means' denoising performance (average PSNR) on several datasets with the compared implementations (noise standard deviation of level 20).

in color handling, all the experiments in this chapter are grayscale. The denoising performance will be compared on several datasets: BSD68 a set of 68 images from the Berkeley segmentation dataset [MFTM01], Kodak a set of 24 images from the Kodak dataset [Fra99] and IPOL a set of 16 images from [Col14][1]. In addition, we compare the denoising performance and running time for two images: Lena and CMLA (from [BD19]). The former is a 512×512 image, while the latter is 4608×3456. Table 4.3 shows the different OpenCL devices, called accelerators (acc.) later on, considered. We consider a high-end multicore CPU (0), an integrated GPU (1), and middle-range consumer grade GPU (2) and a high-end GPU (3).

NL-means In this section, we compare our GPU implementation to the reference CPU code of [BCM11] and to OpenCV (version 4.1.0 [Bra00]) with both a CPU and a GPU implementation. We compare the denoising performance with additive white Gaussian noise of standard deviation $\sigma = 20$. For this noise level, [BCM11] recommends 5×5 patches and $h = 0.4\sigma = 8$. However, likely due to differences in the implementation choices, $h = 8$ did not give good results for OpenCV. By maximizing the PSNR on a serie of images of the Waterloo dataset [MDW+17], we found the best parameter for OpenCV to be $h = 20$ for this noise level with 5×5 patches. Due to the differences in our implementation (number of neighbors, patch size, flat patch trick, aggregation), a different parameter h was needed as well. We used $h = \sigma$. We compare our implementation for two patch sizes. By default, our implementation uses 8×8 patches with a subgrid step of four. Thus a pixel is covered by four patches. We also show results for patches of size 5×5 and a subgrid step of one, which corresponds to the parameters of the reference implementation.

We illustrate the denoising performance of the implementations on standard datasets in Ta-

[1] Available on http://mcolom.info/download/no_noise_images/no_noise_images.zip

Method	Lena (PSNR-Time)	CMLA (PSNR-Time)
[BCM11]	31.59 - 8.5 $\pm$0.2 s	31.75 - 522 $\pm$5 s
OpenCV	31.20 - 279 $\pm$2 ms	31.12 - 16.0 $\pm$0.3 s
OpenCV GPU	31.18 - 6 $\pm$1 ms	Out of memory
Ours P5,S1	32.05 - 4.27 $\pm$0.02 ms	32.32 - 219.2 $\pm$0.5 ms
Ours	31.86 - 0.699 $\pm$0.007 ms	31.99 - 20.7 $\pm$0.1 ms

Table 4.5: Comparison of NL-means' denoising performance (PSNR) and execution time with the compared implementations. CPU methods were run on a single core of an Intel(R) Xeon(R) W-2145 CPU @ 3.70GHz. GPU methods were run on accelerator 3. The running times and the 95% confidence intervals were estimated from 100 runs. Running time for GPU methods corresponds to the sum of time spent running GPU kernels, obtained with nvprof or OpenCL profiling information. Running time for CPU methods corresponds to execution time after the image is loaded and before the result is saved.

Accelerator	search	filtering	Total
0	1672 $\pm$6 ms	282 $\pm$4 ms	1950 $\pm$7 ms
2	40 $\pm$1 ms	24.0 $\pm$0.3 ms	64.5 $\pm$0.5 ms
3	15.91 $\pm$0.08 ms	4.44 $\pm$0.02 ms	20.7 $\pm$0.1 ms

Table 4.6: Comparison of the time taken on the compared accelerators for the main passes (NL-means with our default parameters) on the CMLA. The running times and the 95% confidence intervals were estimated from 100 runs.

ble 4.4. As can be seen, OpenCV performs about the same with the CPU and GPU backends, but underperforms compared to the reference implementation. Our code with the default parameters (*Ours*) was slightly outperformed by the reference implementation on BSD68 and Kodak, but performed better on the IPOL dataset. When analyzing the results, one can see that our method has better denoising performance on images with flat regions thanks to the flat patch trick. Due to the use of larger patches than recommended for this noise level, the performance is slightly lower elsewhere. When using our code with 5×5 patches and a subgrid step of 1 (*Ours P5, S1*), our method outperforms the reference implementation on all datasets tested. The gains over [BCM11] can be explained by the use of a bilinear scheme for the combination of the results of overlapping patches, instead of using a constant weight, and by the flat patch trick.

In Table 4.5, we compare both the denoising performance (PSNR) and the execution time on Lena and CMLA. One can observe that OpenCV's CPU implementation is more than 30 times faster than the reference code, although at the cost of a lower PSNR. Our GPU code runs significantly faster with the default parameters than with 5×5 patches and a subgrid step of 1, but the denoising quality is slightly lower. Both compared versions of our code ran faster than OpenCV's GPU implementation on Lena, while obtaining better denoising results. OpenCV could not denoise CMLA on the GPU as it required intermediate buffers that would not fit in the 12GB GPU memory. One can see that the execution time of our implementation does not scale linearly with the image resolution. The difference in running time per pixel between both images can be explained by the effects of cache (Lena fits completely in the GPU cache) and better use of the parallelization capabilities of a GPU in the case of CMLA which is bigger. The running time of our implementation on three different accelerator is analyzed in Table 4.6. Most of the running time is spent on the patch search, rather than the NL-means weighting (filtering). The aggregation

Method	patch size	search region size	2D transform	1D transform	patch step	Number of nearest neighbors	Maximum distance for a nearest neighbor
[DFKE07b]	8×8 / 8×8	39×39 / 39×39	Bior/DCT	Haar/Haar	3/3	16/32	2500/400
[Leb12]	8×8 / 8×8	33×33 / 33×33	Bior/DCT	Hadamard/Hadamard	3/3	16/32	2500/400
OpenCV	4×4 / 4×4	16×16 / 16×16	Haar/Haar	Haar/Haar	1/1	8/8	2500/400
[HK19]	8×8 / 8×8	23×23 / 23×23	DCT/DCT	Hadamard/Hadamard	3/3	16/32	3000/400
[WS17]	8×8 / 8×8	32×32 / 32×32	DCT/DCT	DCT/DCT	4/4	8/8	3000/400
Ours	8×8 / 8×8	21×21 / 21×21	Bior/DCT	Hadamard/Hadamard	4/4	8/8	2500/400

Table 4.7: Default parameters for $\sigma = 20$ for all *BM3D* methods compared. Parameters in gray can be modified easily by changing a value in the code, or passing an argument when calling the method.

Method	BSD68	Kodak	IPOL
[Leb12]*	29.32	30.68	33.06
[HK19]*	29.26	30.65	33.05
Ours*	29.27	30.66	33.06
[Leb12]	29.50	30.83	33.23
OpenCV	29.04	30.24	31.82
[HK19]	29.26	30.61	32.94
[WS17]	27.34	29.60	31.94
Ours	29.34	30.65	32.91

Table 4.8: Comparison of denoising performance (average PSNR) of BM3D on several datasets of the compared implementations (noise standard deviation of level 20). *: The default parameters were replaced by the fixed parameters used to compare the three implementations.

of the results (normalization of an aggregation buffer) is small and therefore not displayed. It is nonetheless included in the total running time. Accelerator 1 could not compute the result on CMLA as the execution time was more than the timeout threshold of this GPU. We can notice our code executing on CPU (accelerator 0) is competitive with the compared CPU implementations (the equivalent single core time is 31s, compared to *OpenCV*'s 16s and [BCM11]'s 522s), even though our code was not optimized for CPU specifically. This is likely due to an efficient use of the CPU extended instruction set (SSE, AVX, etc) for the code generated by OpenCL.

BM3D There are many different implementations of BM3D available. We compare our code to two GPU implementations, [HK19, HK18] and [WS17], and two CPU implementations, [Leb12] and OpenCV's. [Leb12] is a reference CPU re-implementation of the original paper and *OpenCV* is the implementation that can be found in OpenCV. We only consider open-source implementations reproducing the original method described in [DFKE07b].

In Table 4.7, we show the default parameters of these methods as well as which parameters can be modified. No other methods use the default parameter proposed in the original BM3D publication. Only our method can be configured to have exactly these parameters. It has to be noted, though, that the Hadamard and Haar transforms have been shown to give equivalent results in [DFKE07b], thus arguably, [Leb12]'s parameters are almost identical to the original, except for the default search region size.

Due to the different sets of 3D filters supported by each implementation, they are not exactly

Method	Lena (PSNR-Time)	CMLA (PSNR-Time)
[Leb12]*	32.98 - 7.68 ±0.06 s	33.39 - 491 ±5 s
[HK19]*	32.97 - 22.24 ±0.07 ms	33.40 - 986.5 ±0.8 ms
Ours*	32.98 - 16.15 ±0.05 ms	33.40 - 513 ±4 ms
[Leb12]	33.04 - 6.84 ±0.08 s	33.38 - 425 ±3 s
OpenCV	31.87 - 3.36 ±0.07 s	31.71 - 204.6 ±0.5 s
[HK19]	32.94 - 17.97 ±0.04 ms	33.07 - 760.7 ±0.7 ms
[WS17]	32.11 - 52 ±4 ms	32.67 - 5.06 ±0.01 s
Ours	32.71 - 2.59 ±0.02 ms	32.89 - 91.9 ±0.5 ms

Table 4.9: Comparison of denoising performance (PSNR) and execution time between for the compared BM3D implementations. CPU methods were run on a single core of a Intel(R) Xeon(R) W-2145 CPU @ 3.70GHz. GPU methods were run on accelerator 3. The running times and the 95% confidence intervals were estimated from 100 runs. Running time for GPU methods corresponds to the sum of time spent running GPU kernels, obtained with nvprof or OpenCL profiling information. Running time for CPU methods corresponds to execution time after the image is loaded and before the result is saved. *: The default parameters were replaced by the fixed parameters used to compare the three implementations.

Acc.	search	filtering	search	filtering	Total
0	446 ±5	2106 ±9	5230 ±10	2150 ±10	9930 ±30
1	1090 ±10	962 ±4	1081 ±5	713 ±4	3860 ±10
2	17.6 ±0.2	87.2 ±0.2	30.2 ±0.2	105.0 ±0.2	241.9 ±0.2
2*	17.6 ±0.3	35.3 ±0.4	20.2 ±0.3	38.3 ±0.3	124.6 ±0.5
3	6.47 ±0.04	33.34 ±0.07	11.75 ±0.05	39.61 ±0.07	91.8 ±0.2

Table 4.10: Comparison of the time taken (in ms) on the compared accelerators for the main passes (BM3D with our default parameters) on CMLA. The running times and the 95% confidence intervals were estimated from 100 runs. *: Using long atomics instead of float atomics

comparable. Thus, we only compare using the same parameters, [Leb12], [HK19] and our implementation. The patch search region is set to 39×39, the 2D transforms are both set to DCT, and the 1D transforms to Hadamard. The patch step is 3, and the maximum number of nearest neighbors is 16 for the first pass, and 32 for the second pass.

We also compare the performance of all implementations with their default parameters. Similarly as before, we show the denoising performance on standard datasets in Table 4.8 and denoising performance and running time for two images in Table 4.9. Details about the running time of our method can be found in Table 4.10. As we can see, *OpenCV* and [WS17] underperform compared to other methods. This is likely due to their choice of 2D or 1D transforms.

The three compared methods set to the same fixed parameters had similar denoising performance, which validates the equivalence of these implementations, besides the default parameters. However our implementation was faster than the other GPU implementation [HK19]. The main optimization compared to [HK19] is the use of a single kernel to implement all the filtering operations (see Section 4.4.4). When comparing the performance of all the implementations with their default parameters, [Leb12] had the best denoised results, and our implementation was the fastest. Our default parameters were chosen to maximize denoising performance while looking for aggressive speedups. As a result, it ran almost nine times faster than the closest competitor. However as our parameters can easily be changed, our code can also be used to reproduce the results of the best performing [Leb12]. This is done at a cost similar to the one reported in Table 4.9 as the main difference between the parameters suggested in [DFKE07b] and the fixed one used for the table is the use of Bior instead of DCT for the first pass.

Table 4.10, which analyzes the running time of our method on several accelerators, shows that, contrary to NL-means, most of the running time is spent in the filtering of the patch stacks, rather than in the search of neighbors. One limiting factor of this filtering is the writing of the results on an accumulation buffer with emulated float atomics. As said in Section 7, hardware long atomics can be used if the range of the input is known. We can see in Table 4.10 that accelerator 2 benefits significantly from hardware long atomics since the total running time is almost divided by two. The hardware of accelerator 3 is able to do hardware float atomics and long atomics (available in CUDA), but these functionalities are not available for the OpenCL driver at the time of writing. Both implementations [HK19] and [WS17], written in CUDA, used hardware float atomics. We notice that, like it was for NL-means, our BM3D code running on accelerator 0 (CPU) is competitive with the compared CPU implementations. It runs at an equivalent single-core execution time of 159s, compared to 205s for *OpenCV* and 425s for [Leb12]. One has to be careful though when comparing these results because the parameters are different.

VBM3D and video NL-means In this section, we delve into the performance of the video variations of the previous algorithms.

While NL-means has been previously used for video denoising, there is no standard video version. Thus, for this comparison, we use our code extending the search to a 3D window including the 4 past images and the 4 future images of the current sequence (similarly to VBM3D). In addition, we use 16×16 patches. The use of bigger patches is justified for video denoising as it results in better matches of the same content in the previous frames, which improves the denoising result and the temporal consistency.

We compared with VBM3D using the original parameters of [DFE07] as implemented by [EA20], and our implementation (which has a different patch size for the second pass, as explained in 4.2.3). To highlight the improvements of the video algorithms over the image versions, we show the denoising performance of NL-means and BM3D with our default parameters, used separately on each frame (no 3D search window).

Table 4.11 shows the denoising performance and average running time per frame on the Derf

Method	SF	GPU	PSNR	Time/frame
NLM ours	yes	yes	31.54	0.83 $\pm$0.06 ms
BM3D ours	yes	yes	32.93	4.7 $\pm$0.3 ms
NLM video ours	no	yes	33.53	10.0 $\pm$0.3 ms
VBM3D [EA20]	no	no	34.21	3.0 $\pm$0.2 s
VBM3D ours	no	yes	34.31	3.4 $\pm$0.2 ms

Table 4.11: Comparison of denoising performance (average PSNR) and average running time per frame of VBM3D and NL-means on the Derf dataset of the compared implementations (noise standard deviation of level 20). Whether the method uses a single frame (SF) or a GPU is indicated.

Acc.	search	filtering	search	filtering	Total
0	22.5 $\pm$0.7	31.5 $\pm$0.6	48 $\pm$1	72 $\pm$1	174 $\pm$4
1	28 $\pm$1	20.5 $\pm$0.8	65 $\pm$2	50 $\pm$2	164 $\pm$8
2	1.7 $\pm$0.1	1.13 $\pm$0.04	3.3 $\pm$0.2	2.6 $\pm$0.3	9.0 $\pm$0.5
3	0.54 $\pm$0.07	0.55 $\pm$0.05	0.90 $\pm$0.02	1.1 $\pm$0.1	3.4 $\pm$0.2

Table 4.12: Comparison of the average time per frame taken (in ms) on the compared accelerators for the main passes for our VBM3D implementation (with our default params) on the Derf dataset.

dataset of the compared implementations. The Derf dataset corresponds to the Derf's Test Media collection[2] as processed in [AFM18a]: The original videos are RGB of size 1920×1080, and seven grayscale sequences of 100 frames were extracted and down-sampled by a factor two (the resolution is thus 960×540).

Video methods obtain a significantly superior PSNR (a 1.99 db gain for NL-means and 1.28db for BM3D). While NL-means is slower in its video version than its image version, due to the bigger patch search region, VBM3D is faster than our BM3D with default parameters. This is due to the competitive video patch search algorithm of VBM3D, and due to the first pass of the algorithm using a patch step of 6 for VBM3D, while our BM3D uses 4. The fact that our GPU implementation has a slightly better denoising performance (of 0.10 db) compared to the reference CPU implementation is due to the use of 8×8 patches in the second pass. This phenomenon was already observed in Table 4.2, on the same dataset (with a different noise generation). For VBM3D, Table 4.12 shows that computation time is quite balanced between all steps even though the search takes slightly more time than the filtering. It's also important to notice that the implementation achieves real-time denoising on these sequences for both accelerators 2 and 3.

4.6 Conclusion

In this chapter, we proposed a GPU implementation of NL-means, BM3D and VBM3D. The technical challenges of this endeavor and of the proposed solutions have been discussed. The PSNR denoising performance was shown to match, and sometimes even to surpass thanks to careful implementation choices, the denoising performance of the reference CPU implementations while being several orders of magnitude faster. We also proposed compromises for the default parame-

[2] https://media.xiph.org/video/derf

ters to get an even larger speedup at a minor cost for the denoising performance. This work paves the way to new uses of these algorithms in time constrained environments.

Part II

Learning-based video and burst denoising

5 Non-Local video denoising by CNN

Non-local patch based methods were until recently the state of the art for image denoising but are now outperformed by CNNs. In video denoising however, they are still competitive with CNNs, as they can effectively exploit the video temporal redundancy, which is a key factor to attain high denoising performance. The problem is that CNN architectures are not compatible with the search for self-similarities. In this chapter we propose a simple, yet efficient way to feed video self-similarities to a CNN. The non-locality is incorporated into the network via a first non-trainable layer which finds for each patch in the input image its most similar patches in a search region. The central values of these patches are then gathered in a feature vector which is assigned to each image pixel. This information is presented to a CNN which is trained to predict the clean image. We apply the proposed method to image and video denoising. In the case of video, the patches are searched for in a 3D spatio-temporal volume. The proposed method achieves state-of-the-art performance. This work has been published in [DEM+19] and [DEM+20].

5.1 Introduction

Recently, we have seen denoising methods moving away from patch-based methods presented in Part I and a new trend using machine learning for denoising emerging.

Image denoising has a vast literature where a variety of methods have been applied: PDEs and variational methods (including Markov random fields, MRF, models) [ROF92a, CCC+10, Rot05], transform domain methods [DJ94], non-local (or patch-based) methods [BCM05b, DFKE07b], multiscale approaches [FPM17b], etc. See [LCBM12] for a review. In the last two or three years, CNNs have taken over the state of the art. In addition to attaining better results, CNNs are amenable to efficient parallelization on GPUs potentially enabling real-time performance. We can distinguish two types of CNN approaches to image denoising: *trainable inference networks* and *black box* networks.

In the first type, the architecture mimics the operations performed by a few iterations of optimization algorithms used for MAP inference with MRFs prior models. Some approaches are based on the Fields-of-Experts model [Rot05], such as [Bar09, SR14, CP17]. The architecture of [VTL16] is based on EPLL [ZW11], which models the *a priori* distribution of image patches as a Gaussian mixture model. Trainable inference networks reflect the operations of an optimization algorithm, which leads in some cases to unusual architectures, and to some restrictions in the network design. For example, in the *trainable nonlinear reaction diffusion network* (TNRD) of [CP17] even layers must be an image (i.e. have only one feature). As pointed out in [KKHP17] these architectures have strong similarities with the residual networks of [HZRS16].

The black-box approaches treat denoising as a standard regression problem, not using much of the domain knowledge acquired during decades of research in denoising. The first denoising approaches using neural networks were proposed in the mid and late 2000s. Jain and Seung [JS09] proposed a five layer CNN with 5×5 filters, with 24 features in the hidden layers and sigmoid activation functions. Burger et al. [BSH12] reported the first results competitive with patch-based methods using a multilayer perceptron trained to denoise 17×17 patches, but with a heavy architecture. More recently, DnCNN [ZZC+17b] obtained impressive results with a far lighter 17 layer deep CNN with 3×3 convolutions, ReLU activations and batch normalization [IS15]. This work also proposes a blind denoising network that can denoise an image with an unknown noise level $\sigma \in [0, 55]$, and a multi-noise network trained to denoise blindly three types of noise. A faster version of DnCNN, named FFDNet, was proposed in [ZZZ18], which also allows handling spatially varying noise by adding a noise map $\sigma(x)$ as an additional input. The architectures of DnCNN and FFDNet keep the same image size throughout the network. Other architectures [MSY16, SMD17, CCXK18] use pooling or strided convolutions to downscale the image, and then up-convolutional layers to upscale it back. Skip connections connect the layers before the pooling with the output of the up-convolution to avoid loss of spatial resolution. Skip connections are used extensively in [TYLX17].

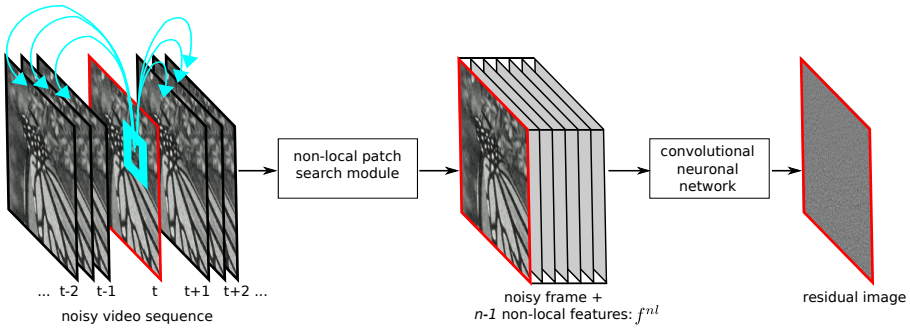


Figure 5.1: Illustration of the proposed *Video Non-Local Network* (VNLnet). The first module performs a patch-wise nearest neighbor search across neighboring frames. Then, the current frame, and the feature vectors f^{nl} of each pixel (the center pixels of the nearest neighbors) are fed into the network.

Although these architectures produce very good results, for textures formed by repetitive patterns non-local patch-based methods still perform better [ZZC+17b, BSH12]. Some works have therefore attempted to incorporate the non-local patch similarity into a CNN framework. Qiao *et al.* [QDF+17] proposed inference networks derived from the non-local FoE MRF model [ST11]. This can be seen as a non-local version of the TNRD network of [CP17]. A different non-local TNRD was introduced by [Lef17]. BM3D-net [YS18] pre-computes for each pixel a stack of similar patches and feeds them into a CNN that reproduces the operations done by (the first step of) the BM3D algorithm: a linear transformation of the group of patches, a non-linear shrinkage function and a second linear transform (the inverse of the first). The linear transformations and the shrinkage function are the trainable parameters. In [CFKE18] the authors propose an iterative approach that can be used to reinforce non-locality to any denoiser. Each iteration consists of the application of the denoiser followed by a non-local filtering step. An inconvenience is that the resulting algorithm requires to iterate the denoising network. Trainable non-local modules have been recently proposed by using differentiable relaxations of the nearest neighbor [LWF+18] and k nearest neighbors [PR18] selection rules, or using Graph CNNs [VFM19], where the graph edges encode the non-local connections between pixels with similar features. These approaches are very interesting as they allow the combination of local and non-local interactions in a trainable way.

CNNs have been successfully applied to several video processing tasks such as deblur-

ring [SDW+17], video frame synthesis [LYT+17] or super-resolution [HWW15, SVB18], but their application to video denoising has been limited so far. In [CSY16] a recurrent architecture is proposed, but the results are below the state of the art. More recently, Tassano *et al.* [TDV19a] proposed DVDnet, a convolutional architecture which processes five consecutive frames to predict the central frame. Each frame is first denoised spatially, and then warped to frame t via an optical flow. The aligned frames are stacked together with the central frame and processed by a "temporal denoising" network. The authors use a non-trainable optical flow, which prevents the network from being trained end-to-end. Two recent works proposed networks without explicit motion estimation: ViDeNN-G [CvG19] processes three consecutive frames, and applies first a spatial denoising followed by temporal denoising, similar to [TDV19a], except that the frames are stacked without aligning them. A different architecture, named fastDVDnet, was proposed in [TDV19b]. Instead of first using a spatial denoising, three consecutive noisy frames are stacked together. The stack is processed by a U-net [RFB15] which predicts the central frame. To extend the temporal receptive field of the network, the authors cascade two levels of these networks. The overall network takes five frames as input. Recently [GMU18b, MBC+18a, EDAF19] focused on the related problem of image burst denoising reporting very good results. There is also recent work focusing on unknown noise-model denoising for videos that use self-supervision for training [EDM+19, EDAF19].

Before the advent of CNNs, patch-based methods were the state of the art [DFE07, MBFE12, AM18b, EAM17, BLM16, WLPB17], and some continue to be competitive in terms of denoising performance [AM18b, EAM17], albeit with a larger computational cost). They exploit extensively the self-similarity of natural images and videos, namely the fact that most patches have several similar patches around them (spatially and temporally). Each patch is denoised using these similar patches, which are searched for in a region around it. The search region generally is a space-time cube, but more sophisticated search strategies have also been used. Because of the use of such broad search neighborhoods these methods are called *non-local*. While these video denoising algorithms perform very well, they often are computationally costly.

Patch-based methods usually follow three steps that can be iterated: (1) search for similar patches, (2) denoise the group of similar patches, (3) aggregate the denoised patches to form the denoised frame. VBM3D [DFE07] improves the image denoising algorithm BM3D [DFKE07b] by searching for similar patches in neighboring frames using a "predictive search" strategy which speeds up the search and gives some temporal consistency. VBM4D [MBFE12] generalizes this idea to 3D patches. In VNLB [AM15], video extension of [LBM13c], spatio-temporal patches that were not motion compensated are used to improve the temporal consistency. In [EAM17] a generic search method extends every patch-based denoising algorithm into a global video denoising algorithm by extending the patch search to the entire video. SPTWO [BLM16] and DDVD [BL17] use optical flow to warp the neighboring frames to each target frame. Each patch of the target frame is then denoised using the similar patches in this volume with a Bayesian strategy similar to [LBM13c]. Recently, [WLPB17] proposed to learn an adaptive optimal transform using batches of frames.

Patch-based approaches have also been applied in frame-recursive denoising methods [EMA18, AM19b], that denoise each frame using only the corresponding noisy frame and the previous denoised frame.

It works particularly well in the case of video denoising, where it achieves state-of-the-art performance.

The method first computes for each image patch the n most similar neighbors in a rectangular spatio-temporal search window and gathers their central pixels forming a feature vector which is assigned to each image location. This results in an image with n channels, which is fed to a CNN trained to predict the clean image from this high dimensional vector. We trained our network for

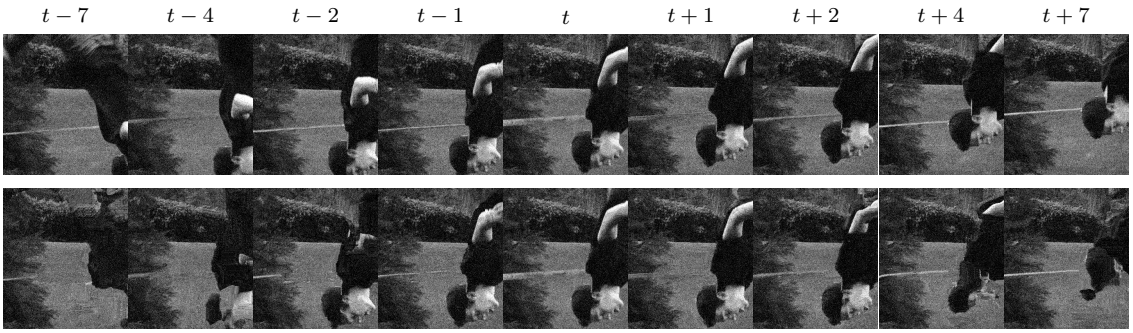


Figure 5.2: Spatio-temporal video crop (top row) and the features extracted by the non-local search for the central frame (bottom row), applying the restriction of one neighbor per frame.

grayscale and color video denoising. Practically, training this architecture is made possible by a GPU implementation of the patch search that allows computing the nearest neighbors efficiently.

To train our network we created a dataset of 17k video segments. In the two testing datasets, our network obtains state-of-the-art results on both color and grayscale video denoising. The code to generate the datasets and reproduce our results is available online[1]. A preliminary version of this work was presented in [DEM+19]. The present version includes an extension to color videos, a detailed comparison with recent works, extended discussions comparing these methods, and new experiments.

5.2 Proposed method

Let u be a grayscale video and $u(x,t)$ denote its value at pixel position x in frame t. We observe v, a noisy version of u contaminated by additive white Gaussian noise:

$$v = u + r, \quad \text{where, } r(x,t) \sim \mathcal{N}(0, \sigma^2). \tag{5.1}$$

Our video denoising network processes the video frame by frame. Before it is fed to the network, each frame is pre-processed by a non-local patch search module which computes a non-local feature vector at each image position. A diagram of the proposed method is shown in Figure 5.1. We call our method VNLnet for *Video Non-Local Network*.

5.2.1 Non-local features

Each frame in the video is pre-processed by the non-local patch search module to produce a 3D tensor f^{nl} of n channels. This is the input to the network. The parameter n is the number of nearest neighbor patches searched for each pixel by this pre-processing module.

Let P be a patch of size $s \times s$ centered at pixel x in frame t. The patches are arranged as vectors with s^2 components. The patch search module computes the L_2 distances between the patch P and the patches in a 3D rectangular search region of size $w_s \times w_s \times w_t$ centered at (x,t). The positions of the n most similar patches are (x_i, t_i), ordered either by increasing distance or by increasing frame index t_i.

The pixel values at those positions are gathered to produce the n-dimensional non-local feature vector associated to pixel (x,t):

$$f^{\text{nl}}(x,t) = [v(x_1,t_1), ..., v(x_n,t_n)]. \tag{5.2}$$

[1]The code to reproduce our results, the training and testing datasets can be found at https://github.com/axeldavy/vnlnet.

The non-local features correspond to n images which resemble the frame $v(\cdot, t)$ (see Figure 5.2). One of them is $v(\cdot, t)$ itself, as the reference patch P is always among the n nearest neighbors. The remaining $n - 1$ images are built from the central pixels of the most similar patches to each patch in $v(\cdot, t)$.

One neighbor per frame. We will also consider a restricted version of the patch search, not allowing more than one match per-frame. In this variant, we set the number of matches n to be equal to the number of frames in the search region, resulting in a match for each frame. The neighbors are sorted by frame index instead of the patch distance. With this configuration, the ith non-local feature map corresponds to a warped version of the ith neighboring frame, aligned to the reference frame t. An example is shown in Figure 5.2, for $n = 15$. This can be related to [TDV19a, XCW+19], which align the input frames using an optical flow. Indeed patch matching can be seen as a rough optical flow with integer displacements.

Other alternatives. The proposed non-local features are inspired by classical patch-based methods such as [BCM05c]. Selecting similar patches with the L_2 distance and extracting their central pixel is possibly the simplest way of providing the network with a non-local context. The L_2 distance is widely used, and was shown to be optimal for additive white Gaussian noise (AWGN) [DDT12]. The same work also shows that other patch distances should be used for other noise distributions.

We considered some alternatives to feeding only the central pixels to the network: such as providing as well the patch similarities (which could be used by the network to ignore bad matches), or extracting a small patch (3×3 or 5×5) around the center of every matching patch, instead of the central pixel. However, we did not observe a significant improvement to justify the increase in complexity: the differences in the validation PSNR for these variants were within a 0.1dB range. The reason for this is probably that the added pixels in the patch around the central pixel are redundant with the central pixels of the similar patches for neighboring pixels.

In a previous version of our work [DEM+19] we also included four 1×1 convolutional layers as a trainable transformation of the non-local features. We removed them in this work as we found that the same performance can be achieved with a simpler design.

5.2.2 Network architecture

For processing the non-local features, we considered standard architectures used in image restoration modifying their input layer to the n channel input tensor. We have tested three such architectures.

DnCNN. The DnCNN architecture was proposed in [ZZC+17b] for still image denoising. It consists of a feed-forward network with 17 layers with 64 3×3 convolution kernels, each one followed by batch normalization and ReLU activations. The output layer is a 3×3 convolution. The network outputs a residual image, which has to be subtracted to the noisy image to get the denoised one.

U-Net. U-Nets (also called hourglass networks) have been introduced in [RFB15] for image segmentation. This architecture consists of an encoder-decoder convolutional network. The encoder part downscales the input by a series of pooling layers or strided convolutions, whereas the decoder upscales the coarse features to the desired output resolution. The differentiating aspect of U-Nets from previous encoder-decoder networks are skip connections that bypass each coarser scale. These networks have been applied to many tasks, among them denoising of both single

images [CCXK18, SMD17] and videos [TDV19a]. In our experiments, we use the U-Nets blocks of [TDV19b], except that the first two layers, which feature group convolutions and a number of kernels computed from the number of input frames, are replaced by two standard convolution layers of 32 kernels.

EDSR. EDSR [LSK+17] is a residual CNN architecture tuned for super-resolution. The main architectural differences with DnCNN are the introduction of skip connections every two convolution layers, and the removal of the batchnorm layers. Thus in our experiments we took our DnCNN networks and applied these two changes (the number of convolutional layers and their parameters are kept).

5.3 Datasets and training

We denote by $\mathcal{F}$ the application of the network. The input to $\mathcal{F}$ at time t is $f_t^{\text{nl}} = f^{\text{nl}}(\cdot, t)$, the n-channel image with non-local features gathered from a window of frames around t. The training loss is the mean square error (MSE) between the reconstructed frame and the ground truth clean frame:

$$l(\mathcal{F}(f_t^{\text{nl}}), u_t) = ||\mathcal{F}(f_t^{\text{nl}}) - u_t||_2^2. \tag{5.3}$$

We use residual training, as in [ZZC+17b]. This means that the network actually predicts the noise, and therefore the denoised image is obtained by subtracting the predicted noise from the noisy input.

For RGB videos, we compute the patch search on grayscale frame resulting from averaging the color channels. To form the non-local features we take the RGB components of the central pixels of the matching patches, resulting in a input tensor with $3n$ channels. For RGB videos we only trained a DnCNN network, with 25 layers and 96 channels (i.e. about three times the number of parameters used for grayscale video).

5.3.1 Datasets

For the training and validation sets we used a database of short segments of 16 frames extracted from YouTube videos. Only HD videos with Creative Commons license were used. From each video we extracted several segments, separated by at least $10s$. In total the database consists of 16950 segments extracted from 1068 videos, organized in 64 categories (such as antelope, cars, factory, etc.). As the original videos might contain compression artifacts, noise, etc, we downscaled the video to a height of 540 pixels. This removes the minor artifacts of the videos and better represents clean targets. In addition, we randomized the anti-aliasing filter width (Gaussian blur) of the downscaling. This results in a variety of sharpness/blur in the training dataset, and thus helps reducing dataset bias. We separated 6% of the videos of the database for the validation (one video for each category). For grayscale networks, grayscale data is obtained by converting the previous color datasets.

For training we ignored the first and last frames of each segment for which the 3D patch search window did not fit in the video. During validation we only considered the central frame of each sequence. The resulting validation score is thus computed on 503 sequences (1 frame each).

For testing, we used two distinct datasets. The first one is the dataset of [AFM18b], which was extracted from the Derf's Test Media collection.[2] It is composed of seven sequences of 100 frames of size 960×540. In this dataset, the camera is either still or has a smooth motion. The second one is the `test-dev` split of the DAVIS video segmentation challenge [PTPC+17]. It consists

[2] https://media.xiph.org/video/derf

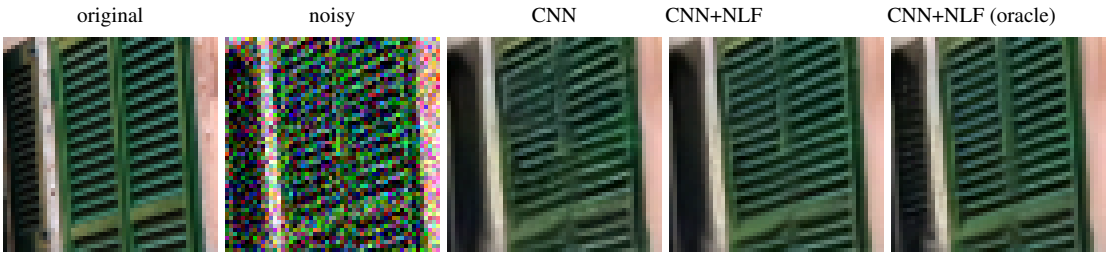


Figure 5.3: Results on still image denoising (AWGN with $\sigma = 25$). Original clean image, noisy image, result obtained with the baseline CNN, result of incorporating the non-local features by finding the nearest neighbors on the noisy image or the oracle noise-less image. Contrast has been linearly scaled for better visualization.

of 30 videos having between 25 and 90 frames. In this dataset, the motion is more challenging. In order to remove compression artifacts and noise present in the original images, both datasets were obtained with a similar downscaling as for the training set (the original images ranged between HD and 4K). Each dataset was processed using a different anti-aliasing filter width.

5.3.2 Training details

At each training epoch, first a subset of the videos of the dataset is selected and noise is added to generate noisy samples. Second the non-local patch search module is run on every video selected. This results in *videos of non-local features* where each frame has n channels containing the output of the patch search module. Third the network is trained on mini-batches built from small crops extracted at random positions on the videos of non-local features.

During training, we ignore spatio-temporal border effects by excluding the first and last $w_t/2$ frames and ignoring crops at borders. At testing time, we simply extended the video by mirroring it at the start and the end of the sequence and adding black borders for the patch-search module.

The training epochs comprise 17000 mini-batches of size 64 square crops of 44 pixels width. We trained for 30 epochs using a combination of recent new optimization techniques which give small performance improvements on various machine learning tasks: Lookahead [ZLBH19], RAdam [LJH+19] and Gradient Centralization [YHHZ20].[3] In addition, we reduced the learning rate at epochs 15 and 27 (from $1e^{-3}$ to $1e^{-4}$ and $1e^{-6}$ respectively). Training a network takes about 14 hours on a NVIDIA TITAN V for grayscale videos, and 24 hours for color videos.

5.4 Experiments and parameter tuning

In this section we evaluate the effect of the non-local features first in still image denoising, and then, after studying the impact of the parameters, in video denoising. Unless otherwise stated, the results reported were obtained using a DnCNN architecture for the fusion network. The network is trained from scratch for each different parameter setting.

5.4.1 The potential of non-locality

Although the focus of this work is in video denoising, it is still interesting to study the performance of the proposed non-local CNN on a single image. Figure 5.3 shows a comparison between a baseline CNN (a 15 layer DnCNN [ZZC+17b]) and a version of our method trained for still

[3] We used the implementation of https://github.com/lessw2020/Ranger-Deep-Learning-Optimizer

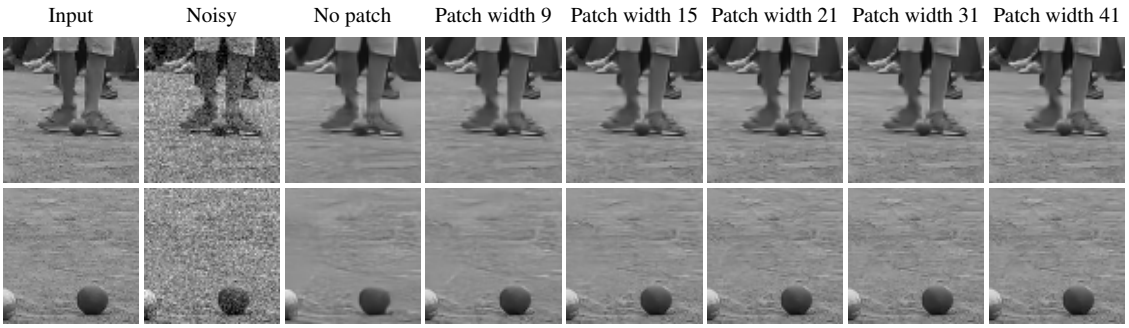


Figure 5.4: Example of denoised results with our method when changing the patch size, respectively no patch search, 9×9, 15×15, 21×21, 31×31 and 41×41 patches. The 3D search window has 15 frames for these experiments.

image denoising (it collects 9 neighbors by comparing 9×9 patches and uses a 15 layer DnCNN architecture. The non-local features are sorted by patch distance. The results with and without non-local information are very similar. The only differences are visible on very self-similar parts like the shutters in Figure 5.3. This is confirmed by quantitative results. The average PSNR on the CBSD68 dataset [MFTM01, ZZC+17b] (noise with $\sigma = 25$) obtained for the baseline CNN is of 31.24dB. The non-local CNN only leads to a 0.04dB improvement (31.28dB). The figure also shows the result of an oracular method: the nearest neighbor search is performed on the noise-free image, though the pixel values are taken from the noisy image. The method obtains an average PSNR of 31.85dB, 0.6dB over the baseline. The oracular results show that non-locality has a great potential to improve the results of CNNs. However, this improvement is hindered by the difficulty of finding accurate matches in the presence of noise. A way to reduce the matching errors is to use larger patches. But on images, larger patches have fewer matches. In contrast, as we will see below, the temporal redundancy of videos allows using very large patches.

5.4.2 Parameter tuning for video denoising

The non-local search has three main parameters: The patch size, the number of retained matches and the size of in the search region (both spatially and in number of frames). We expect the best matches to be past or future versions of the current patch, so we set the number of matches as the number of frames on which we search. In the following we study the impact of the different parameters. Note that for each different parameter, we retrain the fusion network to make sure that it is the optimal one for the chosen parameters. In all cases, we consider denoising of grayscale videos with $\sigma = 20$.

Patch size. In Table 5.1a, we explore the impact of the patch size used for the matching. For each patch size we show results with the unconstrained patch search and with the restriction of one neighbor per frame (ONPF). In both cases, the results improve with the patch size, even for patches much larger than the ones typically used in patch-based methods. The ONPF restriction produces a slight improvement in performance, particularly for larger patches. Figure 5.4 shows visual results obtained with the ONPF restriction. As the patch size grows, there is a noticeable increase in the amount of details recovered from the background.

Number of frames in the search region. In Table 5.1b and Figure 5.5, we see the impact of the number of frames used in the search window (and thus the number of nearest neighbors). One can see that the more frames, the better. Increasing the number of frames beyond 15 (7 past,

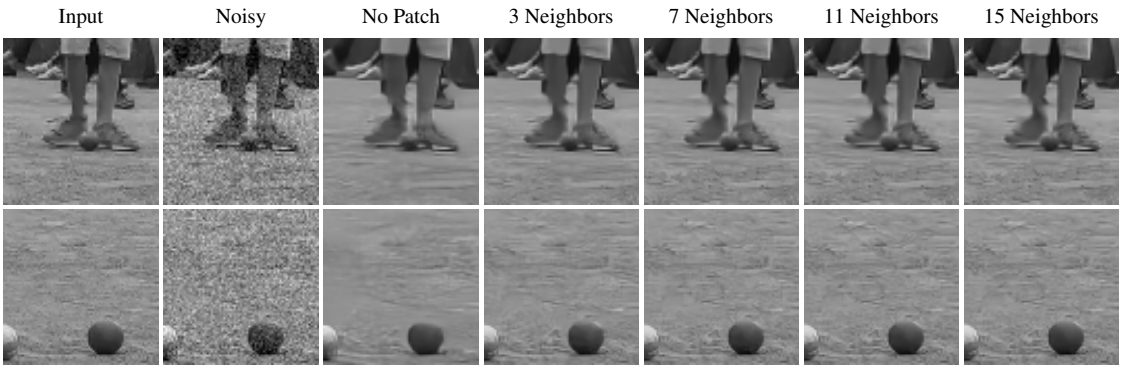


Figure 5.5: Example of denoised results with our method when changing the number of frames considered in the 3D search window (respectively no patch search, 3, 7, 11 and 15). 41×41 patches were used for these experiments.

Patch size	no patch	9×9	15×15	21×21	31×31	41×41
ONPF	33.88	35.70	36.62	37.07	37.39	37.51
free	33.88	35.68	36.55	36.95	37.20	37.32

(a) Impact of the patch size (with 15 frames)

# search frames	no patch	3	7	11	15
ONPF	33.88	35.85	36.93	37.27	37.51

(b) Impact of number of frames (patch size is 41×41)

search size	5×5	11×11	21×21	41×41	51×51
ONPF	36.84	37.13	37.35	37.51	37.53

(c) Impact of search region size (patch size is 41×41, with 15 frames, ONPF)

Table 5.1: Effect of patch size, search region size and number of frames (the number of neighbors is kept equal to the number of frames). All values correspond to the PSNR computed on the validation set for noise with $\sigma = 20$. In 5.1a we compare the performance with the one-neighbor-per-frame restriction (ONPF) and without it (free).

# fusion net	DnCNN	U-Net	EDSR
PSNR	37.51	37.42	37.55

Table 5.2: Impact of the architecture of the CNN used for fusing the features.

# stacked frames	1	3	7	11	15
PSNR	33.88	35.56	36.22	36.42	36.57

Table 5.3: Impact of the number of frames considered for the *video-DnCNN* network (the network input is a 3D crop rather than the result of the non-local search). PSNR on the validation set AWGN with $\sigma = 20$.

current, and 7 future) does not justify the small increase of performance. Foreground moving objects are unlikely to get good neighbors for the selected patch size, unlike background objects, thus it comes to no surprise that the visual quality of the background improves with the number of patches, while foreground moving objects (for example the legs in Figure 5.5) do not improve much.

Spatial size of the search region. Results varying the size of the search region are shown in Table 5.1c. The results improve for larger search regions, although with diminishing returns. From 41×41 to 51×51 there is no significant improvement. Our search region is not compensated by motion, thus having a large search region is necessary to find matching patches on distant frames for moving objects.

Choice of the fusion network. We have evaluated the performance obtained with the two other architectures for the fusion network described in Section 5.2. The results obtained are shown in Table 5.2. The three networks were trained with the same training set and the same procedure. All networks achieve a very similar PSNR, within a 0.13dB range. This shows that the fusion architecture is not critical.

Throughout the rest of the paper, we shall use 41×41 patches and a search volume of 41×41 pixels and 15 frames as the parameters for the patch search. For the fusion network we will use the DnCNN architecture.

5.4.3 Discussion

Surprisingly large patches

An immediate question is why do such big patches achieve such a good performance. On one hand, increasing the patch size has the effect of reducing the variance of the noise in the patch distance, resulting in matches that are less affected by it. This explains the details recovered in the background in Figures 5.4 and 5.5. The large size allow to reliably match the background patches even if the texture seems to be completely obfuscated by noise.

On the other hand, a large patch size becomes a disadvantage for objects with a non-translational motion. In these cases, finding similar matches in the neighboring frames becomes less likely for larger patches. This can be seen in Figure 5.2. The person in the foreground is rotating as she performs a backflip jump. Patches in the foreground object have similar matches only for the closest neighboring frames (e.g. the face cannot be reconstructed for $t \pm 4$), whereas patches in the background can be matched throughout all the temporal extend of the search region. As a consequence, different pixels will have a different number of relevant non-local features. The fusion network learns during training to be robust to these bad matches. It seems to be able to identify when the patch search fails, relying only on the non-local features that are correlated with the target pixel, and adapting accordingly the amounts of spatial denoising and non-local denoising.

This phenomenon can be observed in Figure 5.6, where we compare the results of the proposed VNLnet against the single frame DnCNN from [ZZC+17b] and a *Non-Local Pixel Mean* (NLPM), which simply averages the values of the non-local features given by the patch-matching (please ignore for the moment the column labeled *video-DnCNN*). On the first two rows, the motion is consistent on the whole crop, and thus the output of NLPM is sharp, indicating good matches. As a result, VNLnet's output shows more details than the single frame DnCNN. However the Non-Local Pixel Mean output is blurry for the person in the third row and the background of the fourth row. For these regions, VNLnet and DnCNN both have similar quality. Meanwhile,

Figure 5.6: Example of denoised result for *Non-Local Pixel Mean*, DnCNN, video-DnCNN (see Section 5.4.3) and VNLnet, for AWGN with $\sigma = 20$. The four crops highlight the results on frames feature various kinds of motion. The videos are part of the DAVIS dataset. *Non-Local Pixel Mean* corresponds to the average of the output of the non-local patch search.

for the regions of these two crops with good matches (the background of the third row, and the person of the fourth row), the quality is improved. Using bigger patches will increase the number of patches covering regions of conflicting motion. As a result, we see that the performance gain from 31×31 to 41×41 is rather small.

A note on motion handling

To evaluate the effectiveness of the non-local features we also train a network with the same architecture, but instead of feeding it with the n non-local features we feed it with the stack of the n neighboring frames. We call this network video-DnCNN. We do this for different values of n, and show the results obtained on Table 5.3.

The performance of the video-DnCNN network increases with the number of frames, although less than the proposed VNLnet (see Table 5.1b). In particular, for video-DnCNN the average PSNR on the validation set stagnates at 11 frames. The reason for this is that while the denoising performance increases on sequences with majority of small and smooth motions, it drops significantly when there are many large or irregular motions.

Without the non-local patch-search module, the network has to learn to handle motion implicitly, which makes the task significantly harder. As the number of input frames increases, so does the complexity of the internal motion compensation the network has to learn to denoise accurately. The video-DnCNN network then overfits to the most frequent motion patterns in the training set, and fails when it encounters a different motion. By factoring out the motion, the non-local patch search module removes the need for the network to learn to adapt to various types

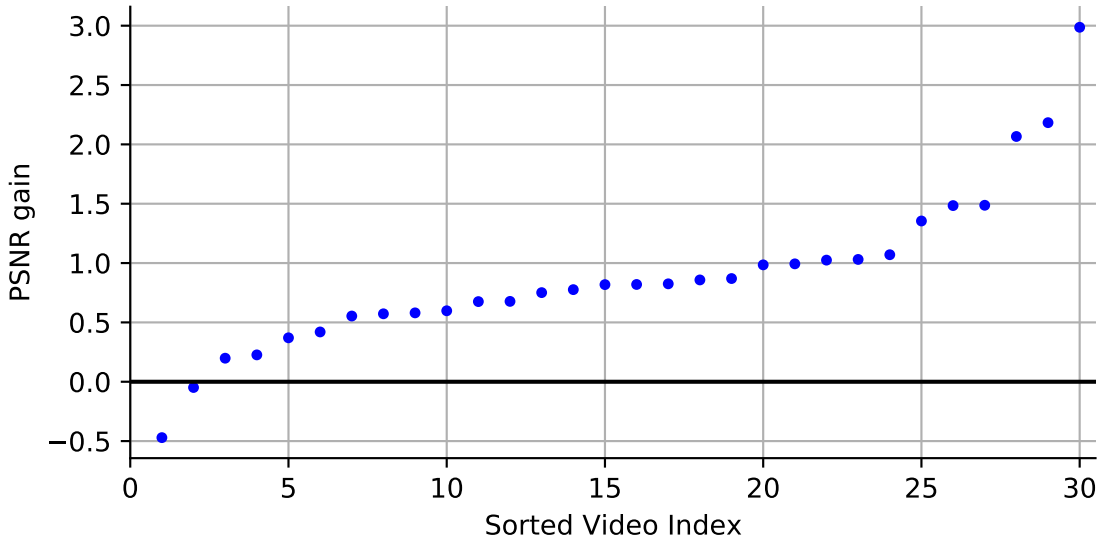


Figure 5.7: PSNR gain of VNLnet versus the network without patch search of Table 5.3 (15 input frames) on grayscale DAVIS ($\sigma = 20$) (35.48db versus 34.59db).

of motion, enabling a better generalization on various moving scenes.

This can be seen in Figure 5.6, by comparing the results of video-DnCNN and VNLnet, for objects with fast/irregular motion patterns. VNLnet is able to recover much more details, thanks to the patch search.

Figure 5.7 shows the PSNR gain of VNLnet over video-DnCNN for each sequence on the grayscale DAVIS test set. The gain given by the non-local patch-search module is significant, except only for two sequences. These feature fixed cameras and static backgrounds covering most of the frame. The sequences with larger gains have complex motion.

Texture is also an important aspect, as patches with a distinct and highly contrasted texture can be matched more reliably. In Figure 5.8 we plot the per patch PSNR gain of VNLnet against the video-DnCNN method and a single frame DnCNN, in terms of a measure of patch "texturedness". We measure patch texturedness as the sum of the squared gradient magnitudes in a patch:

$$T(x,t) = \sum_h \|\nabla u(x+h,t)\|^2 \tag{5.4}$$

where h varies in a patch centered at x, ∇u is spatial gradient of $u(\cdot,t)$ and $\|\cdot\|$ is the Euclidean norm. The texturedness is computed on the clean video. We denoised the grayscale DAVIS test set with $\sigma = 20$, and for each 41×41 patch, we compute its texturedness $T(x,t)$ and the PSNR gain, which we denote by $G(x,t)$. We bin these quantities in a 2D histogram with 400×400 bins which is illustrated by the grayscale image in Figure 5.8 (darker pixels correspond to bins with higher frequency). The figure also shows in red the mean PSNR gain for each level of texturedness, and in blue the same mean, but multiplied by the relative frequency of each texturedness level in the dataset.

The figures confirm that the PSNR gain with respect to both video-DnCNN and single image DnCNN increases with patch texturedness. As expected, the gain is larger with respect to the single frame DnCNN. For patches with very low texturedness the video-DnCNN performs better than the VNLnet. This makes sense, as for those patches the matching will be influenced by the noise. A similar behavior has been observed for patch-based methods in global video denoising [EAM17] and external denoising [MZI13]. Such patches are relatively rare in the test set, which explains the overall positive gains in Figure 5.7.

A related question is whether better results could be obtained replacing the patch search by a frame matching using a standard optical flow. The results of Table 5.1a highlight the importance of very reliable matches, and thus the optical flow would have to be chosen with care. In [TDV19b], the authors of DVDnet [TDV19a] stressed the difficulty of finding a fast and reliable optical flow, and moved away from it for FastDVDnet [TDV19b]. In [XCW+19], the optical flow is computed with a reduced version of SpyNet [RB17], which is trained together with the rest of the network. In our case, the patch search module is not trainable, and the network is trained to process its output.

5.5 Comparison with other methods

In this section, we compare the proposed method VNLnet to VBM3D [DFE07], VNLB [AM15], and DnCNN [ZZC+17b] for grayscale videos, and VBM3D [DFE07], VNLB [AM15], DnCNN [ZZC+17b], ViDeNN-G [CvG19], DVDnet [TDV19a], and FastDVDnet [TDV19b] for color videos. DnCNN was applied frame-by-frame.

We trained grayscale and color networks for AWGN of standard deviation 10, 20 and 40. To highlight the fact that a CNN method can be easily re-targeted to different noise distributions, we also trained a grayscale network for Gaussian noise correlated by a 3×3 box kernel such that the final standard deviation is $\sigma = 20$, and 25% uniform Salt and Pepper noise (removed pixels are replaced by random uniform noise).

Table 5.4 shows the denoising results obtained on the two compared datasets. For grayscale videos, we also include results for SPTWO [BLM16] and VBM4D [MBFE11], computed in [AFM18b] for the DERF dataset. Figures 5.9 and 5.10 show results for the most relevant RGB methods. VNLB (Video Non-Local Bayes) outperformed on average all other methods on the DERF dataset. Meanwhile on the DAVIS dataset, our method performed the best both in grayscale and color, for all the three tested noise levels. VNLB ranked second, except in color for high noise levels, where it was surpassed by DVDnet and FastDVDnet.

A comparison of the results for grayscale and color in Table 5.4 reveals that CNN-based methods exploit better the correlations between color channels: while for grayscale, VBM3D significantly outperforms DnCNN in PSNR on the DAVIS dataset, the reverse occurs for color. Moreover, VNLnet performed proportionally better in color than in grayscale. This should not come as a surprise, since the way in which VBM3D and VNLB treat color is rather heuristic: an orthogonal color transform is applied to the video which is supposed to decorrelate color information. Then the processing of each color channel of a group of patches is done independently.

In order to better compare qualitative aspects of the results we show some details in Figures 5.9 and 5.10. For some scenes, VNLnet recovers significantly more details in the background, as shown in Figure 5.9. In general, we observe that VNLnet, and the other video CNN methods (ViDeNN-G, DVDnet, and FastDVDnet) have better background reconstruction than VNLB. This can be seen in Figure 5.9 and Figure 5.10. Some details however are better recovered by VNLB and VNLnet. For example in Figure 5.10 both methods recover the red lights in the top left corner of the image in the first column, while for the three other methods the lights do not appear. In the second column, VNLB does not reconstruct the trees as well as the CNN-based methods, but manages to recover the color of the clothes of the person in the bottom left. VNLnet not only recovers better the tree structure, but also recovers the clothes correctly. None of the methods restore satisfyingly the grass texture in the third column of Figure 5.10 for the tested noise level. This highlights that there is room for improvement. All five methods achieve reasonable temporal consistency, which is an important quality requirement for video denoising.

One of the benefits of CNNs over traditional model-based approaches is that they can be easily targeted to handle other noise distributions by simply re-training them. We demonstrate this by

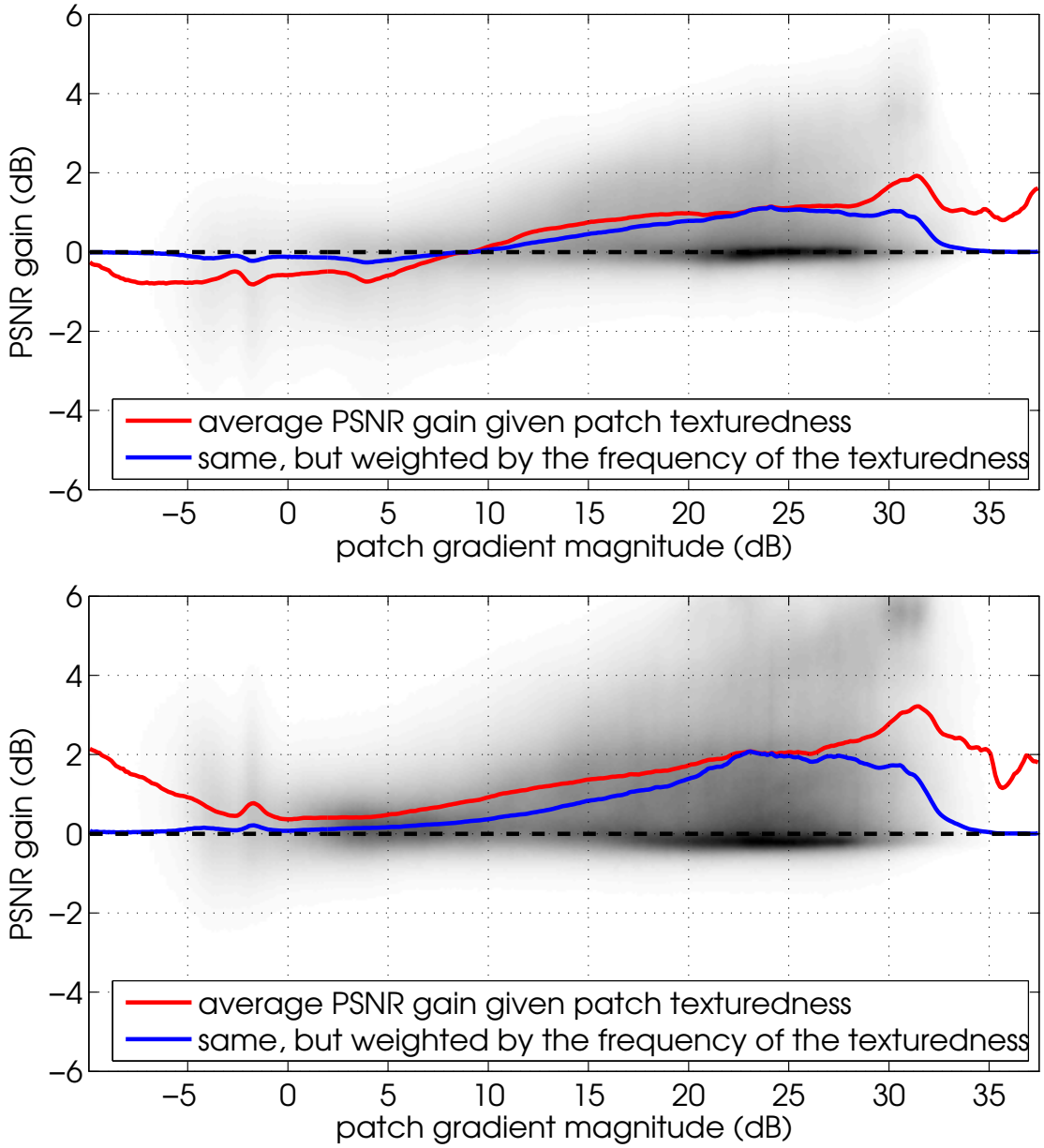


Figure 5.8: Patch PSNR gain between VNLnet versus video-DnCNN (top) and DnCNN (bottom) on grayscale DAVIS with $\sigma = 20$, as a function of a measure of the patch *texturedness* (defined here by the patch gradient magnitude). The grayscale image depicts the 2D histogram of PSNR gains and patch texturedness computed from all 41×41 patches in the test set. The red curve shows the average PSNR gain for each texturedness level, and the blue curve shows the same average PSNR gain, but weighted by the frequency of each texturedness level.

		Method	$\sigma = 10$	$\sigma = 20$	$\sigma = 40$
GRAYSCALE	DERF	SPTWO	39.56 / .9675	35.99 / .9368	30.93 / .7901
		VBM3D	38.24 / .9599	34.68 / .9100	31.11 / .8360
		VBM4D	38.88 / .9534	35.10 / .9169	31.40 / .8432
		VNLB	**40.57 / .9731**	**36.81 / .9428**	**32.95 / .8856**
		DnCNN	37.28 / .9482	33.60 / .8973	30.09 / .8156
		VNLnet	**40.22 / .9730**	**36.51 / .9415**	**32.60 / .8772**
			Corr. Gaussian noise		Uniform S&P 25%
		VNLB	**25.39 / .5922**		**23.49 / .7264**
		VNLnet	**32.92 / .8899**		**48.05 / .9952**
	DAVIS		$\sigma = 10$	$\sigma = 20$	$\sigma = 40$
		VBM3D	37.43 / .9425	33.75 / .8870	30.12 / .8068
		VNLB	**38.84 / .9634**	**35.26 / .9240**	**31.88 / .8622**
		DnCNN	36.80 / .9451	32.94 / .8878	28.69 / .7940
		VNLnet	**39.10 / .9661**	**35.53 / .9305**	**32.03 / .8692**
COLOR	DERF	VBM3D	38.19 / .9560	34.80 / .9165	31.65 / .8568
		VNLB	**40.93 / .9760**	**37.62 / .9528**	**33.97** / .9042
		DnCNN	38.00 / .9588	34.44 / .9171	31.14 / .8520
		ViDeNN-G	38.16 / .9588	35.34 / .9291	32.25 / .8757
		DVDnet	39.08 / .9689	36.48 / .9474	33.43 / **.9051**
		FastDVDnet	39.01 / .9669	36.16 / .9427	33.21 / .9010
		VNLnet	**40.46 / .9748**	**37.36 / .9542**	**33.79 / .9079**
	DAVIS	VBM3D	38.43 / .9591	34.74 / .9157	31.38 / .8473
		VNLB	**40.31 / .9725**	**36.79** / .9420	33.34 / .8896
		DnCNN	38.91 / .9655	35.24 / .9278	31.81 / .8637
		ViDeNN-G	38.46 / .9619	35.47 / .9314	32.32 / .8756
		DVDnet	39.31 / .9702	36.66 / **.9488**	**33.61 / .9059**
		FastDVDnet	39.74 / .9714	36.50 / .9457	33.35 / .9013
		VNLnet	**40.70 / .9760**	**37.32 / .9528**	**33.72 / .9054**

Table 5.4: Quantitative comparison (PSNR and SSIM) of other methods versus the proposed VNLnet on two test sets, both in grayscale and in color. We highlighted the best performance in bold and the second best in bold brown.

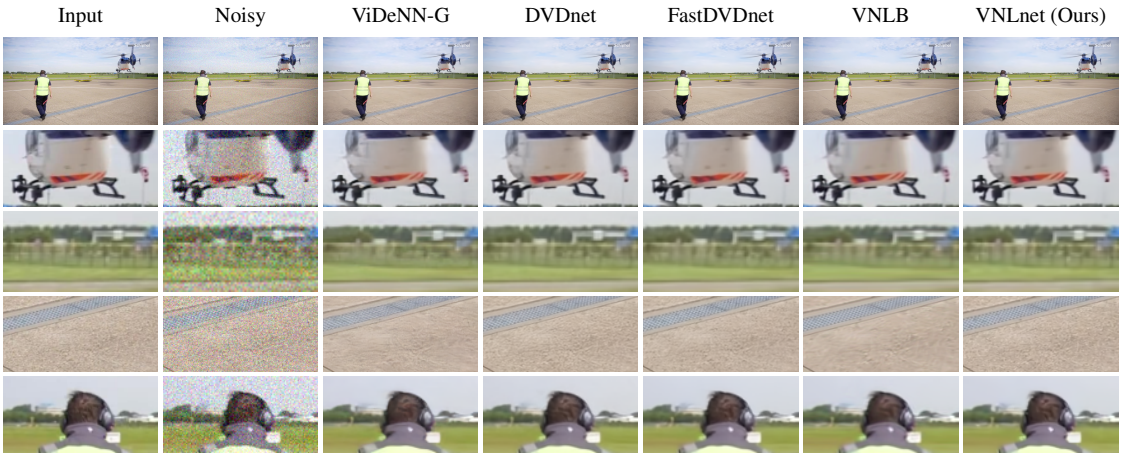


Figure 5.9: Example of denoised result for several algorithms (AWGN with $\sigma = 20$) on a sequence of the color DAVIS dataset [PTPC+17]. The crops highlight the results on non-moving and moving parts of the video.

reporting the results of our method on salt-and-pepper noise and correlated noise in Table 5.4. We include the result of VNLB as a reference.

In summary, the proposed approach for video denoising obtains state-of-the-art results on both test sets. In particular, it outperforms previous CNN approaches. In light of this, we can conclude that effectively exploiting a large number of surrounding frames is key. Indeed, the proposed VNLnet uses 15 frames, in comparison to the 3 frames used by ViDeNN-G and 5 frames of DVDnet and FastDVDnet. Most of these methods avoid relying on an explicit optical flow computation, which can be unreliable given that the input frames are noisy. FastDVDnet and ViDeNN-G do so by performing an early fusion of triplets of consecutive frames without alignment. The non-local features computed via patch correspondences result in an effective way to present the information of large frame neighborhoods to a network that merges them. Training the fusion network is then straightforward.

5.5.1 Running times and number of parameters

In Tables 5.5 and 5.6, we compare the running time in grayscale and color of several methods when denoising a video frame. As before, we consider DnCNN as the fusion network. In Table 5.5, the compared methods are run on a single CPU core, while in Table 5.6, a system with an Intel Xeon W-2145 and a NVIDIA TITAN V is used. For both tables, the loading and writing time of the videos were subtracted. Since we do not have a CPU implementation of the patch search layer, we cannot measure a CPU time for VNLnet. On the GPU, for grayscale videos of 960×540 the non-local search takes 822ms, where as the DnCNN fusion network takes 95ms. Extrapolating this, we could expect a CPU time 8 to 9 times slower than DnCNN.

Most of the running time of our method is spent in the patch search. Our GPU implementation of patch search is similar to the convolution-based patch search described in [DE20]. With the default parameters, the non-local search is costly because matches are searched in 15 frames for patches centered at every pixel of our image. These parameters can be modified, trading off speed by denoising quality. In the Tables 5.7 we show how the time spent in the patch search as a function of the patch size, spatial search region size and number of frames. These behave as expected: the time grows linearly with the number of frames, and quadratically with respect to the patch and search sizes. For example, with a search region of 21×21, the per frame patch search time is reduced by more than three times, while the drop in PSNR is only 0.16dB (see Table 5.1c).

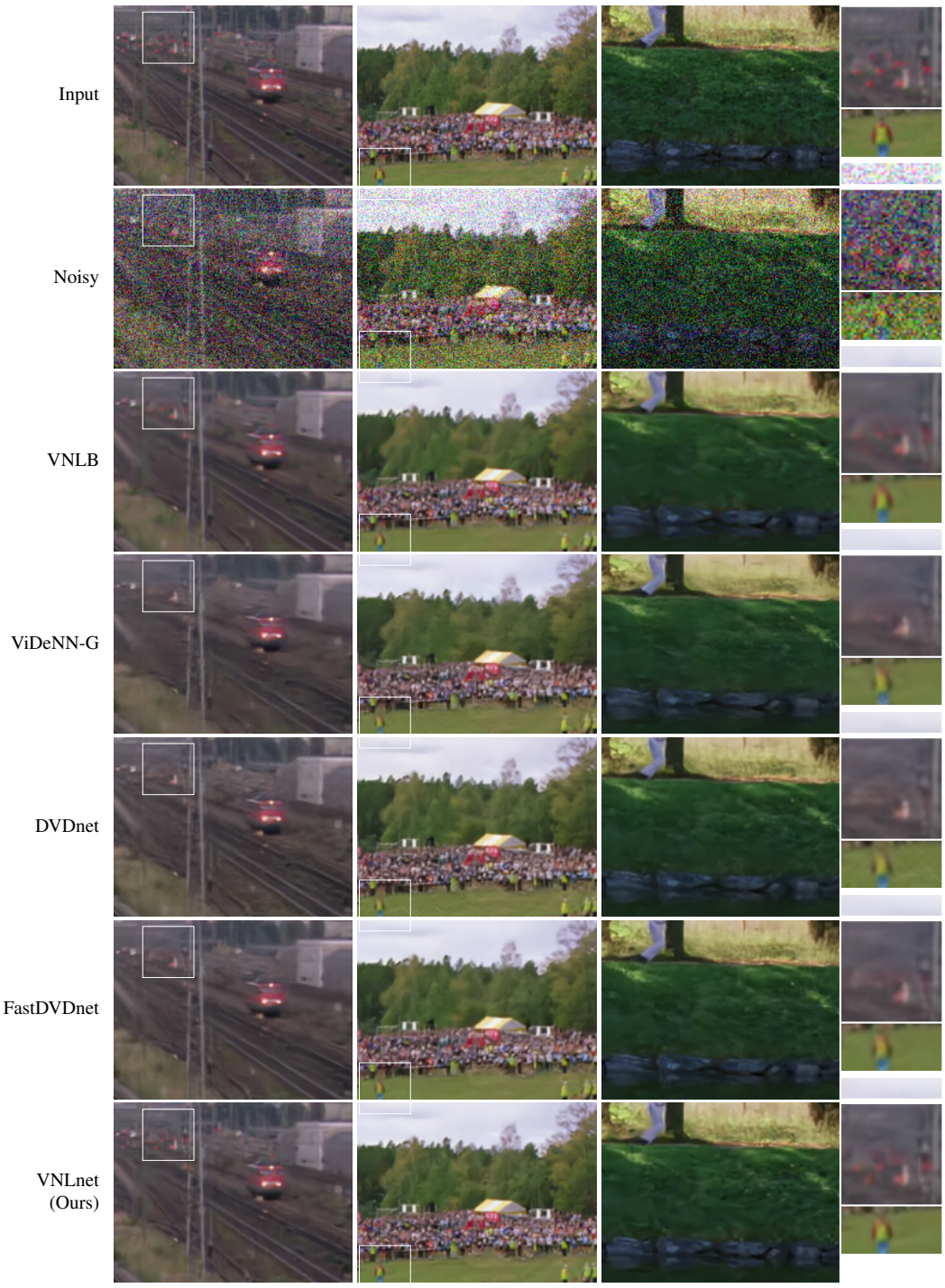


Figure 5.10: Examples of areas where the level of restored detail of the methods differs significantly (AWGN with $\sigma = 40$) in videos of DERF.

VBM3D	DnCNN	VBM4D	VNLB	SPTWO
1.3s	13s	52s	140s	210s

Table 5.5: Running time per frame on a grayscale 960×540 video for VBM3D, DnCNN, VBM4D, VNLB and SPTWO on a single CPU core.

VNLB	ViDeNN-G	DVDnet	FastDVDnet	VNLnet
26.94s	0.186s	2.67s	0.074s	1.16s

Table 5.6: Running time per frame on a color 960×540 video for VNLB, ViDeNN, DVDnet, FastDVDnet, VNLnet on a system with a 16-cores CPU (Intel Xeon W-2145) and a NVIDIA TITAN V.

The table also shows the patch search time normalized by the number of frames that need to be aligned in the search region.

The implementation could be further accelerated by reducing the size of the 3D window using tricks explored in other papers. VBM3D for example centers the search on each frame on small windows around the best matches found in the previous frame. A related acceleration is to use a search strategy based on PatchMatch [BSFG09b].

One of the benefits of the proposed approach is that the fusion network can be simple, resulting in an easier training and a smaller memory footprint of the network weights. The three tested architectures were designed for single image denoising. This reduces the number of trainable parameters, in comparison with other video processing networks. The DnCNN and EDSR networks have both around 0.56M parameters, while the U-Net has 1M. For RGB videos, our DnCNN with 25 layers and 96 channels has 1.9M parameters, roughly the same as, FastDVDnet [TDV19b], which requires 2M parameters as it uses two U-Nets.

Other works such as [XCW+19, TDV19a] use an optical flow sub-network for aligning the frames. This has the advantage that the alignment can be trained together with the fusion network [XCW+19]. However, the smallest current optical flow networks have between 1M and 5M parameters [TD20], and take around 50ms to 100ms (depending on the desired quality) to estimate the optical flow between a pair of frames.

5.6 Conclusions

We described an effective way of incorporating temporal non-local information into a CNN for video denoising. The proposed method computes for each image patch the n most similar neighbors on a spatio-temporal window and gathers their central pixels to form a non-local feature vector which is fed to a CNN. Our method yields a significant gain compared to other CNN approaches. It has similar performance to the best non-CNN method evaluated, VNLB, outperforming it on the largest of our test datasets. In addition, we noted that CNN approaches tend to better reconstruct backgrounds than VNLB, which are perceptually relevant areas. To prevent dataset bias we also proposed a public training set comprising 17k videos from 64 different categories and a simulation strategy that emulates different levels of sharpness.

We have seen the importance of reliable matches: On the validation set, the best performing method used patches of size 41×41 for the patch search. We have also noticed that on regions with non-reliable matches (complex motion), the network seems to revert to a result similar to single image denoising. Thus we believe future works should focus on improving this area.

	9×9	15×15	21×21	31×31	41×41
Per frame	126	201	327	545	822
Normalized	9	14	23	39	59

(a) Average patch search time per frame (in ms) for different patch sizes on a 960×540 video (with a $41 \times 41 \times 15$ search region), and the averaged time normalized by the depth of the search region (minus the central frame).

	3	7	11	15
Per frame	118	346	585	822
Normalized	59	58	59	59

(b) Average patch search time per frame (in ms) for different numbers of frames in the search region on a 960×540 video (patch size is 41×41, search region spatial size is 41×41), and the averaged time normalized by the depth of the search region (minus the central frame).

	5×5	11×11	21×21	41×41	51×51
Per frame	54	100	253	822	1269
Normalized	4	7	18	59	91

(c) Average patch search time per frame (in ms) for different search region spatial sizes on a 960×540 video (patch size is 41×41, with 15 frames), and the averaged time normalized by the depth of the search region (minus the central frame).

Table 5.7: Time spent in the patch search, for different values of the parameters. We report both the average total time spent in the search (considering all frames in the search region) and the time normalized by the depth of the search region (minus the central frame for which there is no search).

6 Model-blind video denoising via frame-to-frame training

In the previous chapters, the noise model is always considered fixed and known. However, modeling the processing chain that has produced a video is a difficult reverse engineering task, even when information on the camera is available. This makes model based video processing still a more complex task. In this chapter we propose a fully blind video denoising method, with two versions batch and online. This is achieved by fine-tuning a pre-trained AWGN image denoising network to the video with a novel frame-to-frame training strategy. Our denoiser can be used without knowledge of the origin of the video and the post-processing steps applied from the camera sensor. The on-line process only requires a couple of frames before achieving visually pleasing results for a wide range of perturbations. It nonetheless reaches state-of-the-art performance for standard Gaussian noise, and can be used offline with the batch fine-tuning for even better performance. This work has been published in [EDM+19].

6.1 Introduction

As we have already seen, a plethora of approaches have been proposed for image and video denoising: PDE and variational methods [ROF92b, CL97], bilateral filters [TM98], domain transform methods [ML99, PSWS03], non-local patch-based methods [BCM05a]. In the last decade, most research focused on modeling image patches [ZW11, YSM12, EA06] or groups of similar patches [DF06, MBP+09, LBM13a, GZZF14, BSH12]. Recently the focus has shifted towards neural networks.

As we have seen in the previous chapters, the most widely adopted assumption in the literature is that noise is additive white Gaussian (AWGN). This is justified by the fact that the noise generated by the photon count process at the imaging sensor can be modeled as Poisson noise, which in turn can be approximated by AWGN after a variance stabilizing transform (VST) [Ans48, MF11b, MF11a]. However, in many practical applications the data available is not the raw data straight from the sensor. The camera output is the result of a processing pipeline, which can include quantization, demosaicking, gamma correction, compression, etc. The noise at the end of the pipeline is spatially correlated and signal dependent, and it is difficult to model. Furthermore the details of the processes undergone by an image or video are usually unknown. To make things even more difficult, a large amount of images and videos are generated by mobile phone applications which apply their own processing of the data (for example compression, filters, or effects selected by the user). The specifics of this processing are unknown, and might

change with different releases.

The literature addressing the case of an unknown noise model is much more limited. The works [LCM15, GPMA18] address denoising noisy compressed images. RF3D [MSMF14] handles correlated noise in infrared videos. Data-driven approaches provide an interesting alternative when modeling is not challenging. CNNs have been applied successfully to denoise images with non-Gaussian noise [ZZC+17a, CCXK18, GYZ+18]. In applications in which the noise type is unknown, one could use *model-blind* networks such as DnCNN-3 [ZZC+17a] trained to denoise several types of noise, or the blind denoiser of [GYZ+18]. These however have two important limitations. First, the performance of such *model-blind* denoising networks very often drops with respect to *model-specific* networks [ZZC+17a]. Second, training the network requires a dataset of images corrupted with each type of noise that we wish to remove (or the ability to generate it synthetically [GYZ+18]). However, generating ground truth data for real photographs is not straightforward [PR17, CCXK18]. Furthermore, in many occasions we do not have access to the camera, and a single image or a video is all that we have.

In this chapter we show that, for certain kinds of noise, in the context of video denoising one video is enough: a network can be trained from a single noisy video by considering the video itself as a dataset. Our approach is inspired by two works: the one-shot object segmentation method [CMPT+17] and the noise-to-noise training proposed in the context of denoising by [LMH+18].

The aim of one-shot learning is to train a classifier network to classify a new class with only a very limited amount of labeled examples. Recently Caelles *et al.* [CMPT+17] suggested a one-shot framework for object segmentation in video, where an object is manually segmented on the first frame and the objective is to segment it in the rest of the frames. Their main contribution is the use of a pre-trained classification network, which is fine-tuned to a manual segmentation of the first frame. This fine-tuned network is then able to segment the object in the rest of the frames. This generalizes the one-shot principle from classification to other types of problems. Borrowing the concept from [CMPT+17], our work can be interpreted as a one-shot blind video denoising method: a network can denoise an unseen noise type by fine-tuning it to a single video. In our case, however, we do not require "labels" (i.e. the ground truth images without noise). Instead, we benefit from the noise-to-noise training proposed by [LMH+18]: a denoising network can be trained by penalizing the loss between the predicted output given a noisy and a second noisy version of the same image, with an independent realization of the noise. We benefit from the temporal redundancy of videos and use the noise-to-noise training between adjacent frames to fine-tune a pre-trained denoising network. That is, the network is trained by minimizing the error between the predicted frame and the past (or future) frame. The noise used to pre-train the network can be very different from the type of noise in the video.

We present the different tools, namely one of the state-of-the-art denoising network DnCNN [ZZC+17a] and a training principle for denoising called noise2noise [LMH+18], necessary to derive our refined model in Section 6.2. We present our truly blind denoising principle in Section 6.3. We compare the quality of our blind denoiser to the state of the art in Section 6.4. Finally we conclude and open new perspectives for this type of denoising in Section 6.5.

6.2 Preliminaries

The proposed model-blind denoiser builds upon DnCNN and the noise-to-noise training. In this section we provide a brief review of these works, plus some other related work.

Figure 6.1: From the *same* starting point and *only* using the video, our fine-tuned network is able to denoise different noises without any artifact. The top images are the noisy and the bottom ones the denoised. From left to right: Gaussian noise, Poisson type noise, salt and pepper type noise and JPEG compressed Gaussian noise.

6.2.1 DnCNN

DnCNN [ZZC+17a] was the first neural network to report a significant improvement over patch-based methods such as BM3D [DF06] and WNNM [GZZF14]. It has a simple architecture consisting of 17 convolutional layers. The first layer consists of 64 3×3 followed by ReLU activations and outputs 64 feature maps. The next 15 layers also compute 64 3×3 convolutions, followed by batch normalization [IS15] and ReLU. The output layer is simply a 3×3 convolutional layer.

To improve training, in addition to the batch normalization layers, DnCNN uses *residual learning*, which means that network is trained to predict the noise in the input image instead of the clean image. The intuition behind this is that if the mapping from the noisy input f to the clean target u is close to the identity function, then it is easier for the network to learn the *residual mapping*, $f \mapsto f - u$.

DnCNN provides state-of-the-art image denoising for Gaussian noise with a rather simple architecture. For this reason we will use it for all our experiments.

6.2.2 Noise-to-noise training

The usual approach for training a neural network for denoising (or other image restoration problems) is to synthesize a degraded image f_i from a clean one u_i according to a noise model. Training is then achieved by minimizing the empirical risk which penalizes the loss between the network prediction $\mathcal{F}_\theta(f_i)$ and the clean target u_i. This method cannot be applied for many practical cases where the noise model is not known. In these settings, noise cannot be synthetically added to a clean image. One can generate noisy data by acquiring it (for example by taking pictures with a camera), but the corresponding clean targets are unknown, or are hard to acquire [CCCY18, PR17].

Lehtinen *et al.* [LMH+18] recently pointed out that for certain types of noise it is possible to train a denoising network from pairs of noisy images (f_i, g_i) corresponding to the same clean underlying data and independent noise realizations, thus eliminating the need for clean data. This allows learning networks for noise that cannot be easily modeled (an appropriate choice of the loss is still necessary though so that the network converges to a good denoising).

Assume that the pairs (f, u) are distributed according to $p(f, u) = p(u|f)p(f)$. For a dataset of infinite size, the empirical risk of an estimator $\mathcal{F}$ converges to the Bayesian risk, i.e. the

expected loss (law of large numbers): $\mathcal{R}(\mathcal{F}) = \mathbb{E}_{f,u}\{\ell(\mathcal{F}(u), f)\}$. The optimal estimator $\mathcal{F}^*$ depends on the choice of the loss. From Bayesian estimation theory [Kay93] we know that:[1]

$$\ell = L_2 \quad \Rightarrow \quad \mathcal{F}^*(f) = \mathbb{E}\{u|f\} \tag{6.1}$$
$$\ell = L_1 \quad \Rightarrow \quad \mathcal{F}^*(f) = \text{median}\{u|f\} \tag{6.2}$$
$$\ell = L_0 \quad \Rightarrow \quad \mathcal{F}^*(f) \approx \text{mode}\{u|f\} \tag{6.3}$$

Here $\mathbb{E}\{u|f\}$ denotes by the expectation of the posterior distribution $p(u|f)$ given the noisy observation f. During training, the network learns to approximate the mapping $f \mapsto F^*(f)$.

The key observation leading to noise-to-noise training is that the same optimal estimators apply when the loss is computed between $\mathcal{F}(f)$ and g, a second noisy version of u. In this case we obtain the respectively the mean, median or mode of the posterior $p(g|f)$. Then, for example if the noise is such that $\mathbb{E}\{g|f\} = \mathbb{E}\{u|f\}$, then the network can be trained by minimizing the MSE loss between $F(f)$ and a second noisy observation g. If the median (resp. the mode) is preserved by the noise, then the L_1 loss (resp. the L_0) loss can be used. The network $\mathcal{F}$ is then trained on pairs of realizations of the same image (x_i^1, x_i^2), using the self-supervised loss $\mathcal{L} = \sum_{(x_i^1, x_i^2)} \|f(x_i^1) - x_i^2\|_p$.

6.3 Model-blind video denoising

In this section we show how one can use a pre-trained denoising network learned for an arbitrary noise and fine-tune it to other target noise types using a single video sequence, attaining the same performance as a network trained specifically for the target noise for classic noise. This fine tuning can be done with batches (using the whole video as a dataset) or online, i.e. frame-by-frame, depending on the application and the computational resources at hand.

As we will show in Section 6.4, starting from a pre-trained network is key for the success of the proposed training, as we do not have a large dataset available as in [LMH+18], but only a single video sequence. The use of a pre-trained network is in part motivated by works on transfer learning such as Zamir *et al.* [ZSS+18]. Denoising different noise models are related tasks. Our intuition is that a part of the network focuses on the noise type while the rest encodes features of natural images.

Our approach is inspired by the one-shot video object segmentation approach of [CMPT+17], where a classification network is fine-tuned using the manually segmented first frame, and then applied to the other frames. As opposed to the segmentation problem, we do not assume that we have a ground truth (clean frames). Instead, we adapt the noise-to-noise training to a single video.

We need pairs of independent noisy observations of the same underlying clean image. For that we take advantage of the temporal redundancy in videos: we consider consecutive frames as observations of the same underlying clean signal transformed by the motion in the scene. To account for the motion we need to estimate it and warp one frame to the other. We estimate the motion using an optical flow. We use the TV-L1 optical flow [ZPB07] with an implementation available in [SPMLF13]. This method is reasonably fast and is quite robust to noise when the flow is computed at a coarser scale.

Let us denote by o_t the optical flow from frame f_t to frame f_{t-1}. The warped f_{t-1} is then $f_{t-1}^w(x) = f_{t-1}(x + o_t(x))$ (we use bilinear interpolation). Similarly, we define the warped clean frame u_{t-1}^w. We assume

(i) that the warped clean frame u_{t-1}^w matches u_t, i.e. $u_t(x) \approx u_{t-1}^w(x)$, and

[1] The median and mode are taken element-wise. For a continuous random variable the L_0-loss is defined as a limit. See [Kay93] and [LMH+18].

(ii) that the noise of consecutive frames is independent.

Occluded pixels in the backward flow from t to $t-1$ do not have a correspondence in frame $t-1$. Nevertheless, the optical flow assigns them a value. We use a simple occlusion detector to eliminate these false correspondences from our loss. A simple way to detect occlusions is to determine regions where the divergence of the optical flow is larger than a threshold τ. This idea and how τ is computed is presented in [BLM16]. We therefore define a binary occlusion mask as

$$\kappa_t(x) = \begin{cases} 0 \text{ if } |\operatorname{div} o_t(x)| > \tau \\ 1 \text{ if } |\operatorname{div} o_t(x)| \leq \tau. \end{cases} \tag{6.4}$$

Pixels with an optical flow that points out of the image domain are considered occluded. In practice, we compute a more conservative occlusion mask by dilating the result of Eq. (6.4).

We then compute the loss masking out occluded pixels. For example, for the L_1 loss we have:

$$\ell_1(f, g, \kappa) = \sum_x \kappa(x) \left| f(x) - g(x) \right|. \tag{6.5}$$

Similarly one can define masked versions of other losses. In the noise-to-noise setting, the choice of the loss depends on the properties of the noise [LMH+18]. The noise types that can be handled by each loss in noise-to-noise have a precise characterization (the mean/median/mode of the *noisy posterior* $p(g|f)$ have to be equal to those of the *clean posterior* $p(u|f)$). Verifying this requires some knowledge about noise distribution. In the absence of such knowledge, since the method is reasonably fast, an alternative would be to test different losses and see which one gives the best result.

In principle our method is able to deal with the same noise types as noise-to-noise. In practice we have some limitation imposed by the registration as it degrades for severe noise. For this reason we do not show examples with non-median preserving noise requiring the L_0 loss. For all our experiments we use the masked L_1 loss since it has better training properties than the L_2 [ZGFK17]. Most relevant noise types often encountered in practice (Poisson, jpeg-compressed, low-freq. noise) can be handled by the L_1 loss and the registration.

We now have pairs of images $(f_t, f_{t_1}^w)$ and the corresponding occlusion masks κ_t and we apply the noise-to-noise principle to fine-tune the network on this dataset. In order to increase the number of training samples the symmetric warping can also be done, i.e. warping f_{t+1} to f_t using the forward optical flow from f_t to f_{t+1}. This allows to double the amount of data used for the fine-tuning. We consider two settings: batch and online training.

Batch fine-tuning. We denote the network as a parametrized function $\mathcal{F}_\theta$, where θ is the parameter vector. In the batch setting we fine-tune the network parameters θ by doing a fixed number N of steps of the minimization of the masked loss over all frames in the video:

$$\theta^{\text{ft}} = \operatorname*{arg\,min}_\theta{}^{N,\theta_0} \sum_{t=1}^{T} \ell_1(\mathcal{F}_\theta(f_t), f_{t-1}^w, \kappa_t) \tag{6.6}$$

where by $\operatorname*{arg\,min}_\theta{}^{N,\theta_0} E(\theta)$ we denote an operator which does N optimization steps of function E starting from θ_0 and following a given optimization algorithm (for instance gradient descent, Adam [KB14], etc.). The initial condition for the optimization is the parameter vector of the pre-trained network. The fine-tuned network is then applied to the rest of the video.

On-line fine-tuning In the on-line setting we train the network in a frame-by-frame fashion. As a consequence we denoise each frame with a different parameter vector θ_t^{ft}. At frame t we compute θ_t^{ft} by doing N optimization steps corresponding to the minimization of the loss between frames t and $t-1$:

$$\theta_t^{\text{ft}} = \underset{\theta}{\overset{N,\theta_{t-1}^{\text{ft}}}{\arg\min}}\, \ell_1(\mathcal{F}_\theta(f_t), f_{t-1}^w, \kappa_t). \tag{6.7}$$

The initial condition for this iteration is given by the fine-tuned parameter vector at the previous frame θ_t^{ft}. The first frame is denoised using the pre-trained network. The fine-tuning starts for the second frame. A reasonable concern is that the network overfits the given realization of the noise and the frame at each step. This is indeed the case if we use a large number of optimization iterations N at a single frame. A similar behavior is reported in [UVL18], which trains a network to minimize the loss on a single data point. We prevent this from happening by using a small number of iterations (e.g. $N = 20$). We have observed that the parameters fine-tuned at t can be applied to denoise any other frame without any significant drop in performance.

The on-line fine-tuning addresses the problem of *lifelong learning* [ZSS+18] by continuously adapting the network to changes in the distribution of noise and signal. This is particularly useful when the statistics of the noise depend on time-varying parameters (such as imaging sensors affected by temperature).

6.4 Experiments

In this section we demonstrate the flexibility of the proposed fine-tuning blind denoising approach with several experimental results. For all these experiments the starting point for the fine-tuning process is a DnCNN network trained for an additive white Gaussian noise of standard deviation $\sigma = 25$. In all cases we use the same hyper-parameters for the fine tuning: a learning rate of 5.10^{-5} and $N = 20$ iterations of the Adam optimizer. For the batch case we use the entire video. The videos used in this section come from Derf's database[2]. They've been converted to grayscale by averaging the three color channels and downscaled by a factor two in each direction to ensure that they contain little to no noise. The code and data to reproduce the results presented in this section are available on `https://github.com/tehret/blind-denoising`.

To the best of our knowledge there is not any other blind video denoising method in the literature. We will compare with state-of-the-art methods on different types of noise. Most methods have been crafted (or trained) for a specific noise model and often a specific noise level. We will also compare with the only available blind denoising method we found, an image denoising method proposed by Lebrun *et al.* [LCM15] which assumes a Gaussian noise model with variance depending on the intensity and the local frequency of the image. This model was proposed for denoising of compressed noisy images. We cannot compare with some more recent blind denoising methods, such as [CCCY18], because there is no code available. We will also compare with DnCNN [ZZC+17a] and VBM3D [DFE07]. VBM3D is a video denoising algorithm. All the other methods are image denoising applied frame-by-frame (perspectives for videos are mentioned in Section 6.5).

The goal of the first experiment is to compare against reference networks trained for these noises the regular way. The per-frame PSNRs are presented in Figure 6.2. We applied the proposed learning process to a sequence contaminated with AWGN with standard deviation $\sigma = 25$, which is precisely the type of noise the network was trained on and verified that it does not deteriorate the pre-training. The batch fine-tuning performs on par with the pre-trained network. The PSNR of the on-line process has a higher variance, with some significant drops for some frames.

[2] https://media.xiph.org/video/derf/

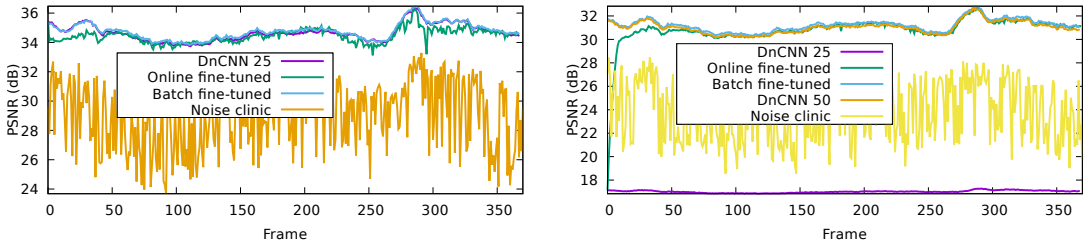


Figure 6.2: The fine-tuning process is done on a sequence corrupted by an additive Gaussian noise of standard deviation $\sigma = 25$ (left) or $\sigma = 50$ (right). The fine-tuned networks (batch and online) achieve comparable performance than the reference networks.

For $\sigma = 50$, we can see that both fine-tuned networks perform better than the pre-trained network for $\sigma = 25$. In fact their performance is as good as the DnCNN network trained specifically for $\sigma = 50$ (actually the batch trained performs even slightly better than the reference network). Our process also outperforms the "noise clinic" of [LCM15].

We have also tested the proposed fine-tuning on four other types of noise: multiplicative Gaussian, correlated, salt and pepper and compressed Gaussian. We present the corresponding per-frame PSNRs in Figure 6.3. The multiplicative Gaussian noise is given by

$$f_t(x) = u_t(x) + r_t(x)u_t(x), \tag{6.8}$$

where $r_t(x)$ is white Gaussian noise with standard deviation of $75/255$ (the images are within the range [0,1]). The resulting variance $\sigma_t^2(x)$ depends on the pixel intensity $u_t(x)$. The correlated noise is obtained by convolving AWGN with a disk kernel. The resulting standard deviation is $\sigma = 25$. The salt and pepper uniform noise is like the one used [LMH+18], obtained by replacing with probability 0.25 the value of a pixel with a value sampled uniformly in $[0, 1]$. Finally, the compressed Gaussian noise, results from compressing an image corrupted by an AWGN of $\sigma = 25$ with JPEG. The last one is particularly interesting because it is a realistic use case for which the noise model is quite hard to estimate [GPMA18]. While in this case the noise can be generated synthetically for training a network over a dataset, this is not possible with other compression tools (for example for proprietary technologies). We can see the effectiveness of the fine-tuning for all examples. The batch training is more stable (smaller variance) and gives slightly better results, although the difference is small.

A visual comparison with other methods is shown in Figure 6.4 for JPEG compressed noise and in Figure 6.5 for AWGN with $\sigma = 50$. Visual examples on real data are presented in the supplementary material. The result of the fine-tuned network has no visible artifacts and is visually pleasing even though the network has never seen this type of noise before the fine-tuning. A limitation of the method is the oversmoothing of texture. Indeed DnCNN has a tendency of oversmoothing textures. Using a network designed for video denoising should help recover more texture and improve temporal consistency [DEM+19]. Another cause is the optical flow. Since it is computed on downscaled noisy frames it is imprecise around edges. This leads to false correspondences between frames and introduces some blur. Improved registration should lead to sharper results.

In Tables 6.1 and 6.2 we show the PSNR of the results obtained on 4 sequences for AWGN of $\sigma = 50$ and JPEG compressed AWGN of $\sigma = 25$ and compression factor 10. For the case of AWGN the fine-tuned networks attain the performance of the DnCNN trained for that specific noise. For JPEG compressed Gaussian noise, the batch fine-tuned network is on average $0.3dB$ above the pre-trained network.

Figure 6.6 shows the impact of different parameters of the method. The main parameters of the proposed training are the learning rate and the number of per-frame iterations. Fewer iterations

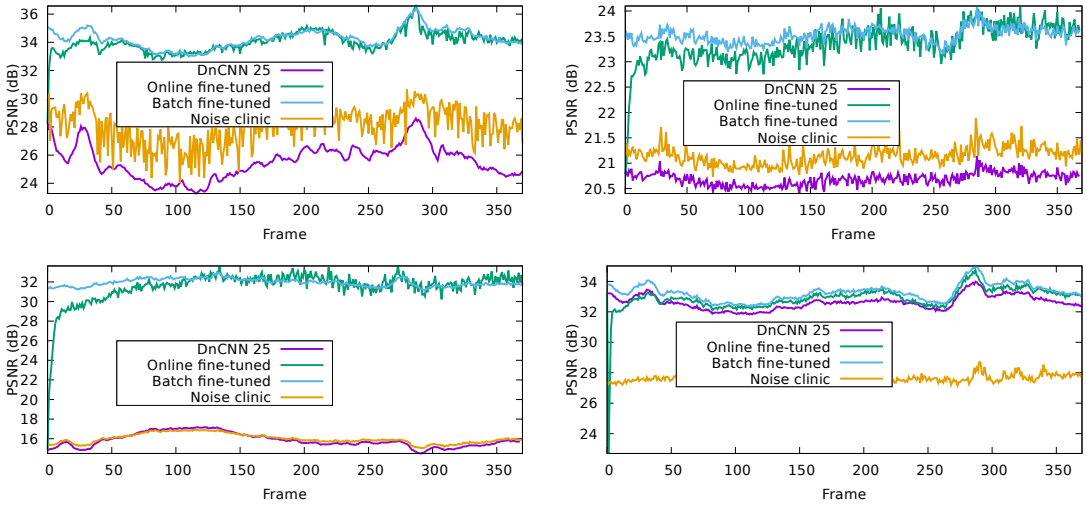


Figure 6.3: Different types of noise. From top-left to bottom right: multiplicative Gaussian noise, correlated Gaussian noise, salt and pepper noise, Gaussian noise after JPEG compression. The fine-tuned network (both online and batch) always performs better than the original network.

Figure 6.4: Example of denoising of an image corrupted by a JPEG compressed Gaussian noise. The fine-tuned network doesn't produce any visible artifacts, contrary to the original DnCNN used for the fine-tuning process. From left to right, top to bottom: Noisy, fine-tuned, VBM3D, ground truth, DnCNN trained for a Gaussian noise, noise clinic.

Figure 6.5: Example of denoising of an image corrupted by a Gaussian noise of standard deviation $\sigma = 50$. The fine-tuned network doesn't produce any visible artifact, the results are comparable to a DnCNN trained for this particular type of noise. From left to right, top to bottom: Noisy, fine-tuned, DnCNN trained for a Gaussian noise with $\sigma = 50$, VBM3D, ground truth, noise clinic, DnCNN trained for a Gaussian noise with $\sigma = 25$.

Method	walk	crowd	football	station	Average
DnCNN 25	17.02	11.24	15.09	13.86	14.30
DnCNN 50	31.02	**25.83**	**31.67**	30.09	29.65
Online fine-tuned	30.84	25.58	31.33	29.90	29.59
Batch fine-tuned	**31.22**	**25.83**	31.54	**30.39**	**29.75**
Noise Clinic	23.85	22.13	24.57	24.39	23.74
VBM3D	*31.57*	*27.02*	*31.97*	*31.33*	*30.47*

Table 6.1: PSNR values for 4 sequences with AGWN of standard deviation $\sigma = 50$.

Method	walk	crowd	football	station	Average
DnCNN 25	32.62	**27.31**	32.48	31.48	30.97
Online fine-tuned	32.86	27.20	32.79	30.88	30.94
Batch fine-tuned	**33.28**	27.19	**32.91**	**31.58**	**31.24**
Noise Clinic	27.62	25.17	27.20	26.89	26.72
VBM3D	*34.16*	*28.95*	*33.83*	*33.53*	*32.62*

Table 6.2: PSNR values on JPEG compressed AWGN noise with $\sigma = 25$ and compression factor 10.

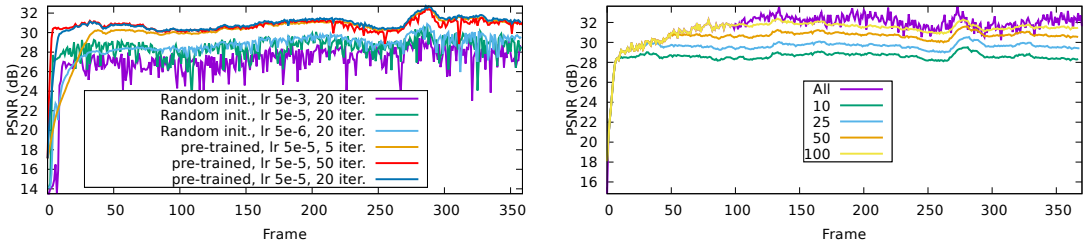


Figure 6.6: Impact of parameters. Left: Impact of the learning rate and the number of iterations. It also shows the gap between using a pre-trained network and random initialization. Right: Impact of the number of frames used for fine-tuning.

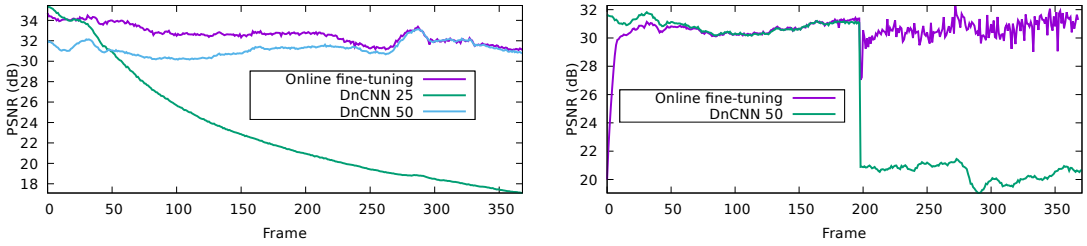


Figure 6.7: Lifelong learning. Left: Slow change. Right: sudden change. The fine-tuned network adapts without difficulty to slow changes and sudden changes. See text for more details

require more frames for convergence. In turn the result has smaller variance. A similar analysis can be done for the learning rate. We also show the importance of using a pre-trained network compared to a random initialization. There is a $2dB$ gap in favor of the pre-trained network. The other important parameter is the number of frames used for fine-tuning. The fine-tuning is stopped at a frame t_0 and $\theta^{\text{ft}}_{t_0}$ is used to process the remaining frame. We can see that the more frames used for the fine-tuning the better the performance.

Finally Figure 6.7 shows examples of lifelong learning. The first example shows a slowly evolving noise (starting with a Gaussian noise with standard deviation $\sigma = 25$ that linearly increases up to $\sigma = 50$). The fine-tuned network performs better than the two reference networks for respectively $\sigma = 25$ and $\sigma = 50$. The second example shows a sudden change (starting with a Gaussian noise with standard deviation $\sigma = 50$ and changes to Salt and pepper noise at frame 200). In that case the fine-tuned network adapts quickly to the new noise model.

The running time depends on the network. We used DnCNN but other networks can be used instead and trained with the proposed method. Each fine-tuning iteration runs a back-propagation step which takes $0.33s$ on a NVIDIA Titan XP for DnCNN for a 960×540 frame. Fifty frames with 20 iterations per frame take 5 mins. For comparison, training DnCNN from scratch over a dataset requires around $6h$ (and a dataset). By using a lighter network and reducing the per-frame-iterations it might be possible to achieve real time frame rates. Moreover, the fine-tuning can be done on a fixed number of frames at the beginning or run in the background each number of frames for cases when the computational efficiency is important.

A challenging test case of our blind denoising is old digitalized films. The difficulty with this data is that the film quality degraded gradually with time and can be physically damaged during its manipulation and reproduction, creating several type of artifacts. The two examples shown in Figures 6.8, 6.9, and 6.10 are examples from footage of World War I [3]. In addition to the noise and the damaged parts of the film, there's also a strong compression that has been applied after digitalization. All of this makes modelling the noise very difficult. Yet, the proposed frame-to-

[3] https://www.army.mil/

Figure 6.8: Blind denoising (on the right) of images coming from a video taken during World War I (original on the left). Blind denoising is useful in this case because it would be nearly impossible to recreate this type of noise to train a network.

frame fine-tuning strategy is still able to learn to denoise these sequences. The blind denoiser is able to remove most of what can be consider as noise while retaining most details. In Figure 6.9 we show a comparison with the pre-trained network (starting point of the fine-tuning) and the result of VBM3D. VBM3D receives as input the noise level σ. We tested several noise levels and chose then one the seemed best. As one can see in Figure 6.9 the blind denoiser keeps more details in the fields, the building or the airplane than the pre-trained network and VBM3D.

Our last experiment is with a video shot with a Samsung Galaxy S7. The video is shot in a low light, and processed by the camera pipeline. This means that it has been demosaicked, denoised (by a fast method running directly on the phone), among other quality enhancement algorithms, and finally compressed. The remaining noise is therefore completely distorted, being colored and non-stationary. We used a pretrained network which is a color DnCNN trained for Gaussian noise with standard deviation $\sigma = 25$. We use these hyper-parameters for the fine tuning: a learning rate of 1.10^{-4} and $N = 10$ iterations of the Adam optimizer. Figure 6.11 presents a crop of the video. Here again, blind denoising largely removes the artifacts left by the phone's pipeline and therefore improves the overall visual quality of the video.

6.5 Discussion and perspectives

Denoising methods based on deep learning often require large datasets to achieve state-of-the-art performance. Lehtinen *et al.* [LMH+18] pointed out that in many cases the clean ground truth images are not necessary, thus simplifying the acquisition of the training datasets. With the framework presented in this chapter we take a step further and show that a single video is often enough, removing the need for a dataset of images. By applying a simple frame-to-frame training on a generic pre-trained network (for example a DnCNN network trained for additive Gaussian noise with fixed standard deviation), we successfully denoise a wide range of different noise models even though the network has *never* seen the video nor the noise model before its fine-tuning. This opens the possibility to easily process data from any unknown origin.

Future works will focus on improving the denoising quality and computation speed. First, given that the application is video denoising, it is expected that better results will be achieved by a video denoising network (the DnCNN network processes each frame independent of the others). Using the temporal information could improve the denoising quality, just like video denoising methods improve over frame-by-frame image denoising methods, but also might stabilize the variance of the result for the on-line fine-tuning. Recent works have also shown the possibility of real time fine-tuning [TRJ+19, TTP+19] paving the way to improve computation time.

Figure 6.9: Blind denoising better preserves details than methods for a predefined noise. From top to bottom, left to right: original, blind denoising, denoising with the pre-trained network and VBM3D with hand-tuned noise parameter.

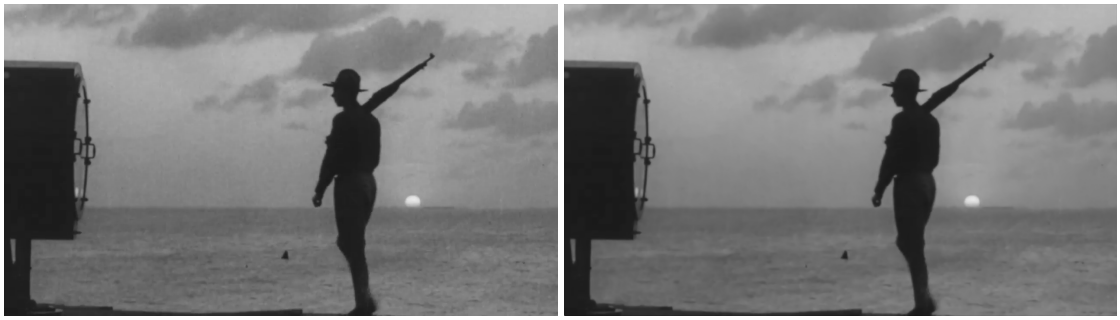


Figure 6.10: Blind denoising (on the right) of images coming from a video taken during World War I (original on the left). Blind denoising is useful in this case because it would be nearly impossible to recreate this type of noise to train a network.

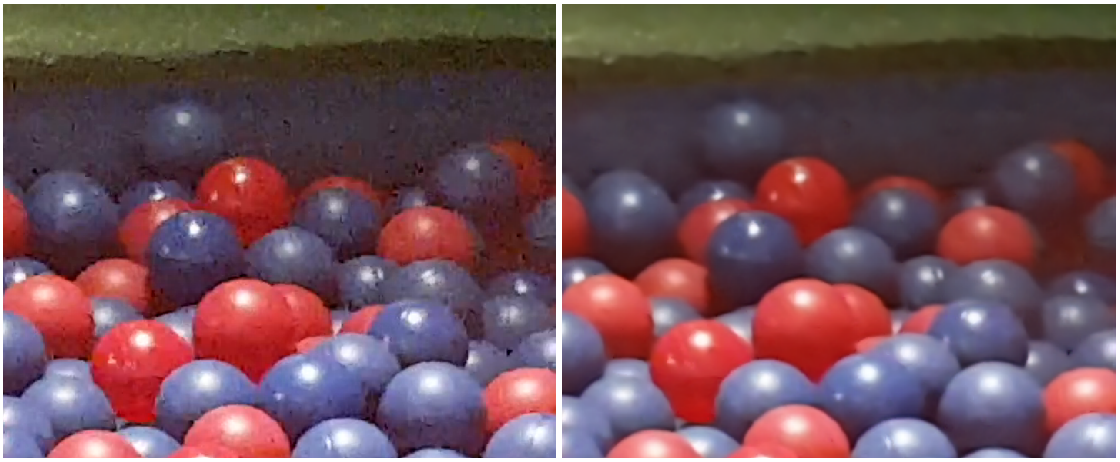


Figure 6.11: Example of denoised image (right) coming from a mobile phone (left). The results is more natural and pleasing to the eye as it doesn't have all these ugly artifacts.

7 Demosaicking with deep learning

Most cameras capture the information of only one color for a given pixel. This results in a mosaicked image that must be interpolated to get three colors at each pixel. The step going from a mosaicked image to a regular RGB image is called demosaicking. This chapter studies two recent demosaicking methods based on convolutional neural networks that achieve artifact-free state-of-the-art results: Deep joint demosaicking and denoising by Gharbi *et al.*and Color image demosaicking via deep residual learning by Tan *et al.*. We show that these methods beat by almost two decibels the best human-crafted methods, while being faster by one order of magnitude. The review presented in this chapter motivates the need for the self-supervised learning presented in the next chapter.

7.1 Introduction

Most cameras capture only one color per pixel, this color being determined by a color filter array (CFA) located on top of the sensor. The most commonly used CFA is the so-called Bayer pattern, consisting of a regular subsampling of each color channel. This means that each pixel of the resulting raw image contains one third of the necessary information, and that the color channels are sampled on different grids. The problem of interpolating the missing color channels at each pixel, by using the information available in a neighborhood, is called demosaicking. It is a challenging ill-posed inverse problem.

The simplest demosaicking method is the independent bilinear interpolation of each channel. Yet recent methods are far more sophisticated. Getreuer [Get12] solves the demosaicking by using contours of objects as guide. Mairal *et al.* [MBP+09] suggested combining group sparsity and dictionary learning to achieve optimal performance. In a series of paper, Kiku *et al.* [KMTO13], [KMTO14] and [MKTO15] proposed increasingly complex interpolation methods where they use the interpolated Green channel as a guide for the Red and Blue channels. These last four methods may be considered among the best ones designed by humans. A recently emerged trend consists in using convolutional neural networks for image processing and computer vision applications. See for example the early attempt in [WL15]. These methods have by now reached state-of-the-art results.

The rest of the chapter is organized as follows: In Section 7.2 we present the method “Deep joint demosaicking and denoising" by Gharbi *et al.* [GCPD16], in Section 7.3 we present the method “Color image demosaicking via deep residual learning" by Tan *et al.* [TZZZ17]. A quantitative and qualitative comparison with previous state-of-the-arts methods is made in Section 7.4.

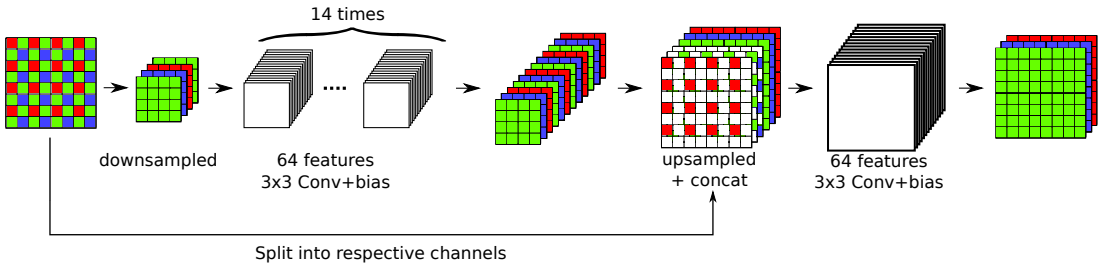


Figure 7.1: Architecture used for demosaicking by Gharbi *et al.* [GCPD16].

Figure 7.2: Mosaicking patterns: Bayer (2×2) and Fujifilm X-Trans (6×6).

7.2 Deep joint demosaicking and denoising by Gharbi *et al.* [GCPD16]

7.2.1 Architecture

The network starts by downsampling the CFA image into a four-channel image that then goes through a serie of 14 Conv+bias layers with 64 features and 3×3 convolutions. A $15th$ layer of Conv+BN+ReLu produces 12 features with 3×3 convolutions. It is followed by an upsampling layer producing an RGB image of twice the width and twice the height. The CFA input is split into the respective channels and concatenated to this intermediate RGB image. This layer acts as a residual layer. After this concatenation, a Conv+bias layer is added before the final layer producing the RGB output. The architecture is depicted in Figure 7.1.

The architecture is slightly modified when working with noisy or Fujifilm X-Trans data. For X-Trans data, the image is not downsampled. Indeed Gharbi *et al.*argue that since the X-Trans pattern is 6×6 (as illustrated in Figure 7.2), the downsampling would be too aggressive. The layer acting as a residual layer is also removed. For noisy data, the noise level (noise standard deviation) σ is concatenated into the downsampling layer (using a channel with values all equal to σ). The rest of the network stays the same.

7.2.2 Training

Loss A classic L_2 loss is used for training in all three cases (Bayer demosaicking, X-Trans demosaicking and joint demosaicking and denoising). All noise levels are trained at the same time for joint demosaicking and denoising.

Training dataset A first pre-training was done using natural images. In particular 1.3 millions images from ImageNet [DDS+09] and 1 million images from MirFlickr [HL08] were chosen for this pre-training. The images chosen had a minimum size of $16M$ pixels and then were downsampled by a factor of 4. Data augmentation was performed by flipping, rotating and applying a 1 pixel-shift. CFA images were obtained by masking.

Gharbi *et al.* argue that classic metrics such as L_2 and PSNR are not much impacted by demosaicking artifacts. Moreover, since difficult patches are rare, this means that it would be

Figure 7.3: Example of images from the luminance dataset (two on the left) and the moiré dataset (two on the right).

hard to train a network for demosaicking without artifacts unless the training set be biased towards the creation of such artifacts. For this reason, they created two datasets (one for each type of artifact, namely luminance and moiré) containing only difficult patches used to fine-tune the network. Using the pre-trained network, luminance artifacts are found using the HDR-VDP2 metric [MKRH11]. Moiré artifacts are found by looking at frequencies that have a much larger energy in Fourier for the demosaicked patch. Figure 7.3 shows examples of patches in these datasets.

Weights initialization He *et al.*suggested in [HZRS15] a way to generalize the *"Xavier"* initialization initially proposed by Glorot and Bengio [GB10] to take into account ReLUs non-linearity. This allows for a good convergence even for very deep models. He *et al.*derive conditions on the weights of the kernels at each layer to avoid the problems of vanishing or exploding gradients and vanishing or exploding signal. It leads to initializing all kernels using a zero-mean Gaussian distribution with variance $\frac{2}{n}$ where n is the size of the kernel.

Training parameters While most parameters were initialized using the method presented in the previous paragraph, biases were initialized to 0. The patch size for the training was set to 128×128 and the batch size to 64. The training was done using the Adam optimizer [KB14]. The learning rate was set to 1.10^{-4} and an L_2 weight decay to 1.10^{-8}. The other parameters for the Adam optimizer were left to their default value. On top of the weight decay, there was a decrease of the learning rate of a factor 10 when the validation error stagnated for ten epochs. The model took about two to three weeks to train on a NVIDIA Titan X[1].

7.3 Color image demosaicking via deep residual learning by Tan *et al.* [TZZZ17]

7.3.1 Architecture

The network starts with a bilinear interpolation of the CFA image into a first-estimate RGB image that then goes through a series of 6 convolutional layers (the three middle layers also have batch-normalization (BN) [IS15] and a ReLU) with 64 features and 3×3 convolutions (convolutions are padded to ensure that the output has the same size as the input). The bilinearly interpolated image is added to the result of these convolutional layers producing a first estimate RGB image. It then goes through a second series of 5 convolutions (the three middle layers also have BN+ReLU) before adding the first estimate to produce the final result. The architecture is depicted in Figure 7.4.

[1]This description of the network training is based on the authors' description. We did not attempt to emulate this training. The neural network weights used here therefore are the original ones provided by the authors.

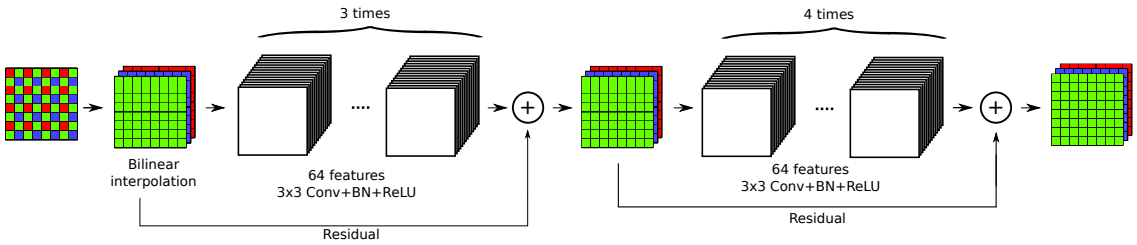


Figure 7.4: Architecture for demosaicking used by Tan *et al.* [TZZZ17].

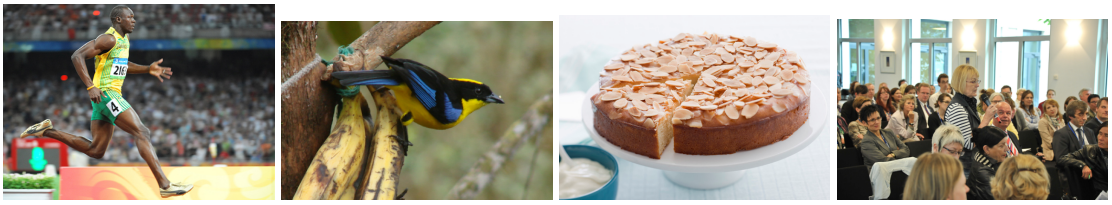


Figure 7.5: Example of images from the Waterloo dataset [MDW+17]

7.3.2 Training

Loss The network can be decomposed into two steps: The first part (corresponding to the first residual section) estimates the green channel and the second estimates the other two channels (using the estimated green channel as a guide). For this reason the loss is split into two terms: the first term focuses only on the green channel and the second one takes into account the final result. For a training pair (x, y), where x is the CFA image and y the corresponding RGB image, and $\theta = (\theta_1, \theta_2)$ the parameters of the network (respectively the first and second part of the network for θ_1 and θ_2, the loss is defined as

$$\mathcal{L}(x, y, \theta = (\theta_1, \theta_2)) = \|y_G - \mathcal{N}_{\theta_1}(x)_G\|_2^2 + \|\mathcal{N}_\theta(x) - y\|_2^2 \tag{7.1}$$

where the G sub-index corresponds to extracting the green channel and $\mathcal{N}_{\theta_1}$ is the partial network up to the first residual section.

Training dataset The dataset used for training is the Waterloo dataset presented in [MDW+17]. A sample of images is shown in Figure 7.5. It is made of 4744 images, out of which 100 are used for validation. Data augmentation was performed by flipping and rotating the images. CFA images were generated by appropriate masking. Finally, out of all training images, 3000 batches of 128 patches were extracted.

Training parameters The network was initialized using the same methods than [GCPD16] presented in Section 7.2.2. The patch size for the training was set to 50×50 and the batch size to 128. The training was done using the Adam optimizer [KB14] with default parameters except for the learning rate. The learning rate was set to 2.10^{-4} for the beginning of the training and 1.10^{-4} for the rest of the training. Early stopping of the training was done as soon as the training error stagnated for more than three epochs. The model took about one day to train on a NVIDIA Titan X[2].

[2]This description of the network training is based on the authors' description. We did not attempt to emulate this training. The neural network weights used here therefore are the original ones provided by the authors.

Method	Kodak	McMaster	Flickr500	All
Gharbi et al. [GCPD16]	41.82	**39.08**	**32.22**	38.54
Tan et al. [TZZZ17]	**41.99**	39.02	32.08	**38.56**
LSSC [MBP+09]	41.44	36.21	29.86	36.86
RI [KMTO13]	39.77	34.76	29.46	35.58
ARI [MKTO15]	39.24	37.48	30.42	36.47
MLRI [KMTO14]	39.85	35.29	29.62	35.83
Getreuer [Get12]	39.30	35.95	29.36	35.74

Table 7.1: Quantitative comparison of different demosaicking methods. The table shows PSNRs on three datasets: the Kodak dataset, the McMaster dataset [ZWBL11] and a subset of the Flickr500 dataset [SCC18].

Method	Gharbi *et al.* [GCPD16]	Tan *et al.* [TZZZ17]	LSSC [MBP+09]	RI [KMTO13]	ARI [MKTO15]	MLRI [KMTO14]	Getreuer [Get12]
Time	13.97	14.06	736.18	1.86	61.32	1.79	3.07

Table 7.2: Computation time of the compared methods. Computation time was computed for the lighthouse image of the Kodak dataset and is given in seconds.

7.4 Quantitative and qualitative comparison

In this section we compare both quantitatively and qualitatively some of the most efficient demosaicking methods. We used the state-of-the-art methods LSSC [MBP+09], RI [KMTO13], ARI [MKTO15], MLRI [KMTO14] and Getreuer [Get12] that are compared in most recent papers and retain a low rank in most. Actually, ARI and LSSC are generally considered the best handcrafted methods, but they are also very complex. Table 7.1 presents the PSNR for all these methods. As one can see the two CNN based methods outperform all previous demosaicking methods. The PSNRs were computed on the Kodak dataset comprised of 24 images, the McMaster dataset from Zhang *et al.* [ZWBL11] comprised of 18 images and a subset of 14 images from the Flickr500 dataset from Syu *et al.* [SCC18]. We also added the average PSNR on a dataset made of the images from all three datasets. In order to avoid boundary effects, the images were first padded (with a padding of 48 pixels) using symmetry before generating the CFA image. This padding was removed before computing PSNRs. Extra care was taken to make sure that the CFA images used for each method were the same. We also used the parameters recommended by the authors for each method. Table 7.2 shows that not only do these methods perform well in terms of PSNR but that they are also reasonably fast. The method [GCPD16] is written in Python and the methods [TZZZ17, MBP+09, KMTO13, KMTO14, MKTO15] are written in Matlab. All computation times were computed on an Intel Core i7-7820HQ even for CNN methods. The CNN based methods were tested without using a GPU. We used the weights provided by the authors for the both CNN methods.

Figures 7.6, 7.7, 7.8 and 7.9 present visual examples of the different demosaicking methods. In particular Figure 7.7 shows that the method with the least moiré artifacts is the method that was trained specifically to avoid this artifact. Figure 7.8 shows that both CNN based methods have no visible zipper effects.

Overall, the two reviewed CNN methods outperform both visually and in terms of PSNR all previous demosaicking methods while also being reasonably fast, even without a GPU.

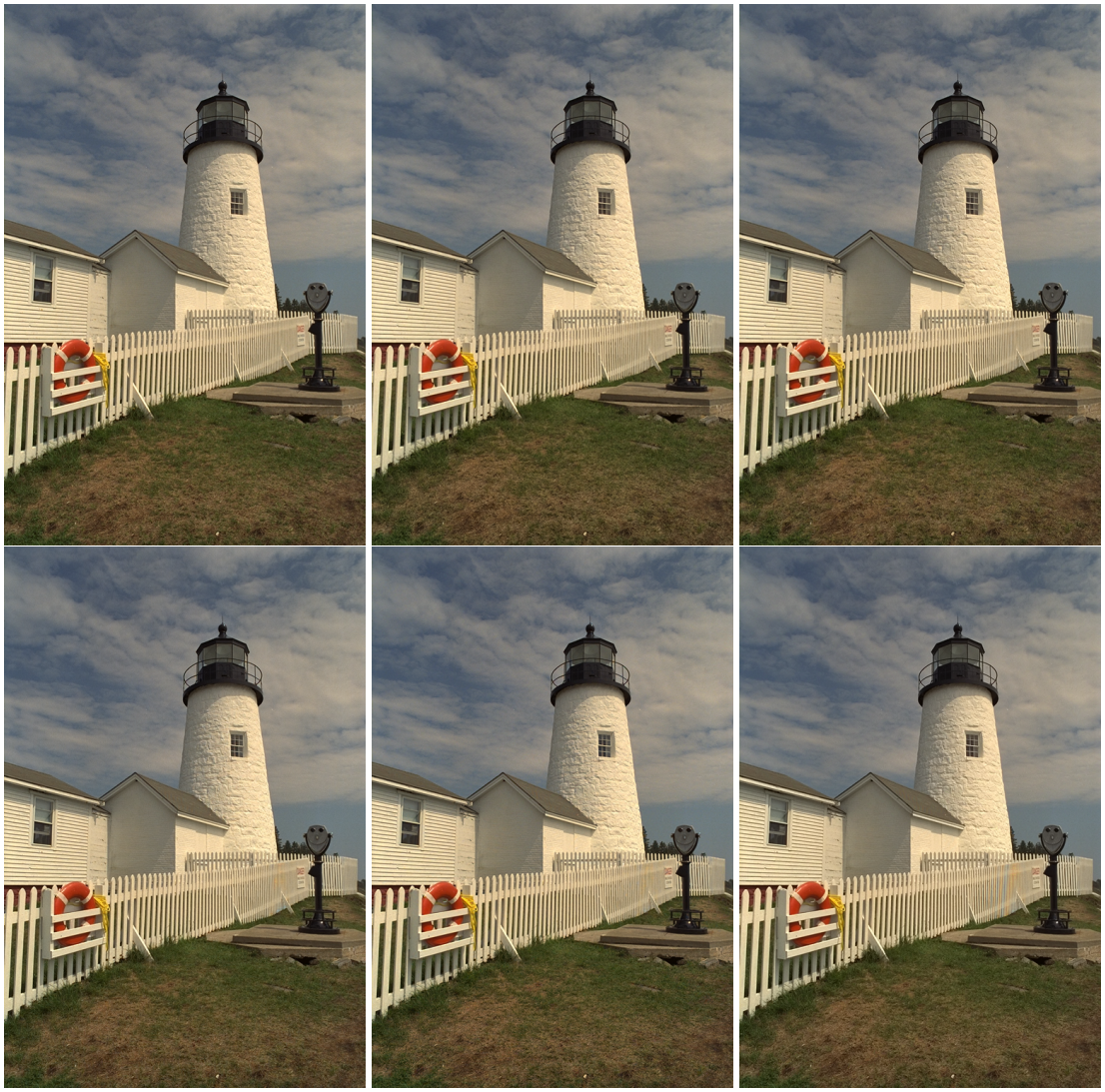


Figure 7.6: From top-left to bottom-right: Original, Gharbi *et al.* [GCPD16], Tan *et al.* [TZZZ17], MLRI [KMTO14], ARI [MKTO15], Getreuer [Get12]. All demosaicking methods shown here perform well overall, with no visible strong artifacts. Yet a moiré artifact appears in the fence for some of the methods.

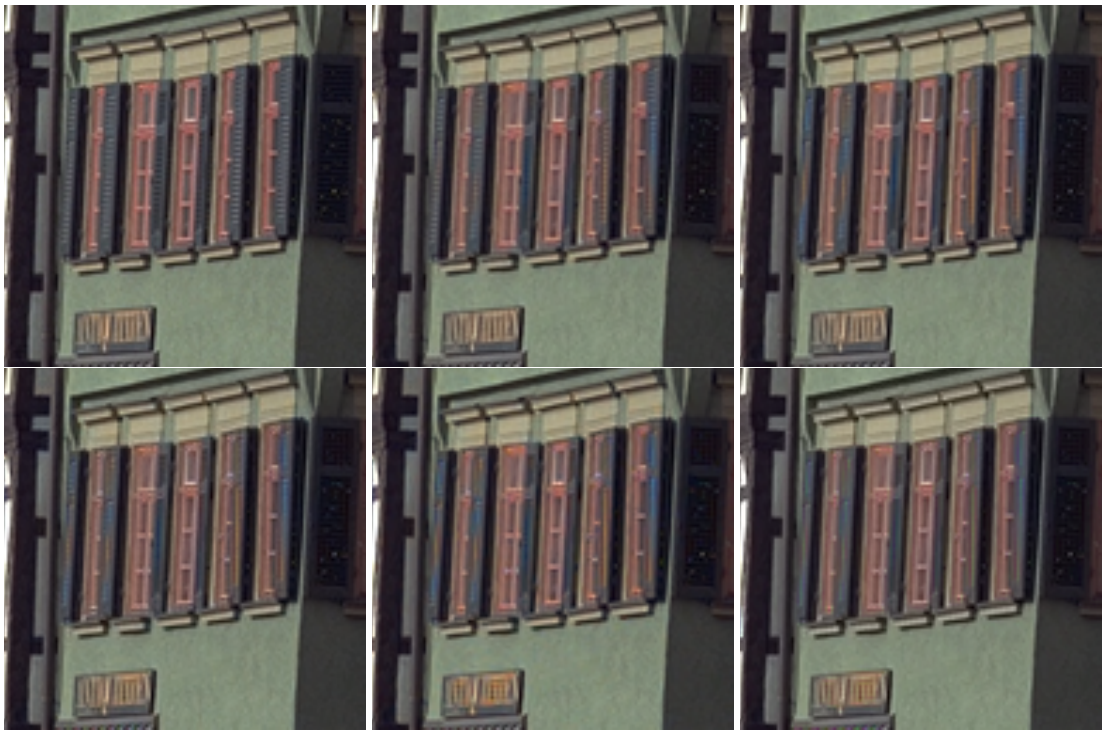


Figure 7.7: From top-left to bottom-right: Original, Gharbi *et al.* [GCPD16], Tan *et al.* [TZZZ17], MLRI [KMTO14], ARI [MKTO15], Getreuer [Get12]. All demosaicking methods still present some moiré artifacts on the shutters. However, the Gharbi *et al.*method, which was trained specifically to avoid this artifact, has fewer than the other methods.

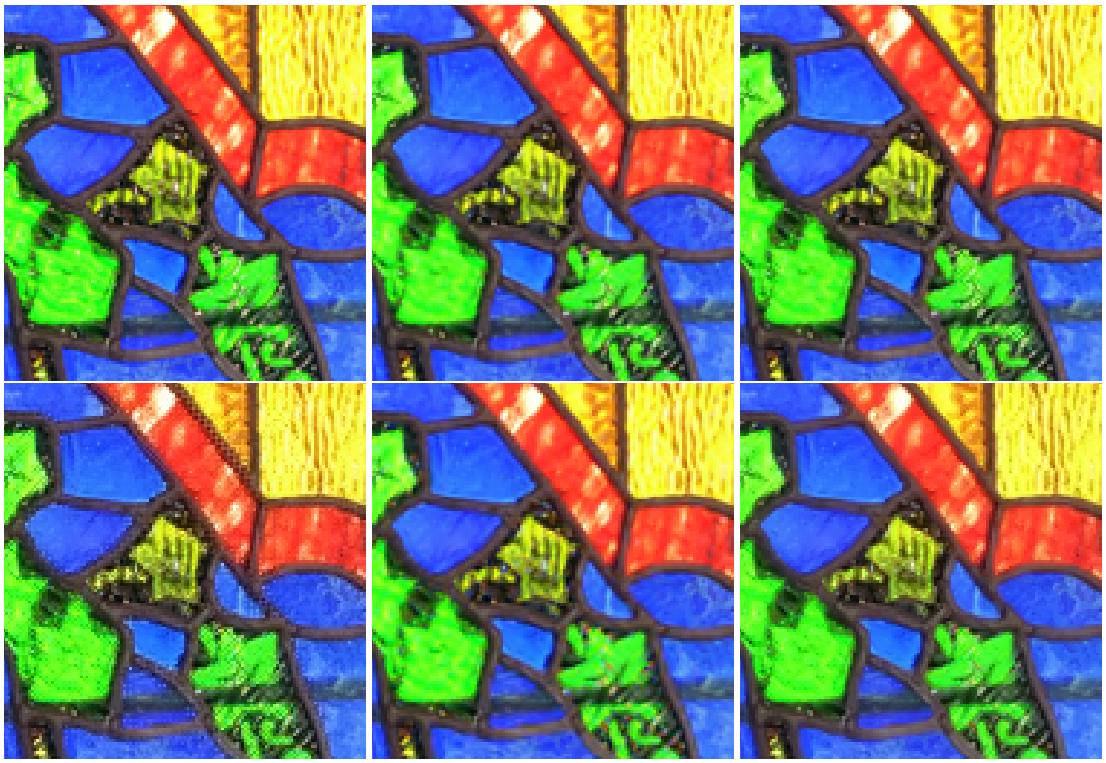


Figure 7.8: From top-left to bottom-right: Original, Gharbi *et al.* [GCPD16], Tan *et al.* [TZZZ17], MLRI [KMTO14], ARI [MKTO15], Getreuer [Get12]. Most methods perform well regarding luminance artifacts. Nevertheless the Gharbi *et al.*method still performs best.

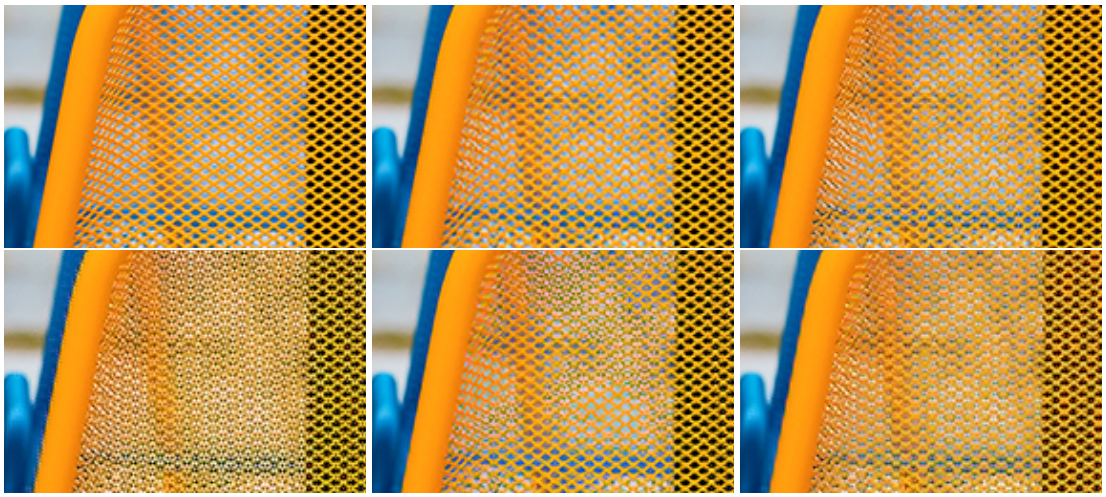


Figure 7.9: From top-left to bottom-right: Original, Gharbi *et al.* [GCPD16], Tan *et al.* [TZZZ17], MLRI [KMTO14], ARI [MKTO15], Getreuer [Get12]. Only the Gharbi *et al.*method performs well visually on really difficult examples.

7.5 Conclusion

We have seen that the CNN based demosaicking methods beat by almost two decibels the best human-crafted methods, while being faster by one order of magnitude. To reach this performance, they did not rely on the clever human techniques established by the anterior state of the art, but simply applied rather standard CNN architectures. This success is explainable. The first reason is that human-crafted algorithms rely on the iteration of nonlinear local filters, the most complex one, ARI, having more than 20 such iterations. But deep convolution networks have the same structure and are in addition scalable in depth and number of filters, until they reach the best performance. Furthermore, human-crafted methods have been designed to avoid certain artifacts, probably to the cost of losing in PSNR. CNNs, on the contrary, can be taught to learn intricate casuistry by the variety of presented of images, and to keep a memory of complex image statistics. Being trained to reach the best PSNR, they definitely reach that goal. Finally, the Gharbi et al. method succeeded in biasing the learning process so as to avoid completely the most annoying moiré and zipper effects, thus achieving the most visually pleasing results of all methods, while still retaining the best PNSR! This shows a clear superiority of deep learning based method over traditional ones and motivates the need for the self-supervised learning presented in the next chapter.

8 Joint demosaicking and denoising by fine-tuning of bursts of raw images

Demosaicking and denoising are the first steps of any camera image processing pipeline and are key for obtaining high quality RGB images. A promising current research trend aims at solving these two problems jointly using convolutional neural networks. Due to the unavailability of ground truth data these networks cannot be currently trained using real RAW images. Instead, they resort to simulated data. In this chapter we present a method to learn demosaicking directly from mosaicked images, without requiring ground truth RGB data. We apply this to learn joint demosaicking and denoising only from RAW images, thus enabling the use of real data. This allows to mitigate the training bias due to simulated data. In addition we show that for this application fine-tuning a network to a specific burst improves the quality of restoration for both demosaicking and denoising. The burst processing is done by fine-tuning a pre-trained network over the pairs of the specific burst. This work has been published in [EDAF19].

8.1 Introduction

As we have seen in Chapter 7, most camera sensors capture a single color at each photoreceptor, determined by a color filter array (CFA) located on top of the sensor. The most commonly used CFA is the so-called Bayer pattern, consisting of a regular subsampling of each color channel. This means, not only that each pixel of the resulting raw image contains one third of the necessary information, but also that the color channels are never sampled at the same positions. The problem of interpolating the missing colors is called demosaicking and is a challenging ill-posed inverse problem. To further complicate things, the captured data is contaminated with noise.

For these reasons the first two steps of a camera processing pipeline are demosaicking and denoising. Traditionally, these problems have been treated separately, but this is suboptimal. Demosaicking first a noisy RAW image correlates the noise making its subsequent denoising harder [PKL+09]. Alternatively, if denoising is applied on the mosaicked data it becomes harder to exploit the cross-color correlations, which are useful for color image denoising [DFKE07a, DVF+09].

Until recently, just like for demosaicking in Chapter 7, state-of-the-art methods for joint demosaicking and denoising were based on carefully crafted heuristics, such as avoiding interpolation across image edges [HP06, PKL+09, ATO15]. Other methods resort to variational principles

Figure 8.1: Using a burst, our fine-tuning (starting from the network from Gharbi *et al.* [GCPD16]) is able to not only denoise well ($\sigma = 5$) but also doesn't show any artifacts like *zipper* or *moire* in the difficult regions. Best visualized on a screen.

where the heuristics are encoded as a prior model [CM12, HEK+14]. In [WGDE+19] both problems are addressed simultaneously by aligning and fusing RAW bursts of frames.

Recent data-driven approaches have significantly outperformed traditional model-based methods [KNJF14, GCPD16, KHKP16, KL18a, KL18b, SCC18]. In [GCPD16], state-of-the-art results are reported with a network trained on a special dataset tailored to demosaicking in which hard cases are over-represented. In [KL18a] an iterative neural network is proposed, later improved by [KL18b] obtaining state-of-the-art performance on both real and synthetic datasets. These networks are relatively lightweight and do not need a lot of training data. The authors in [SCC18] propose two networks for demosaicking. They train on several CFA patterns to compare performance and integrate the handling of denoising with a fine-tuning step. In [ZGFK17] the authors find that the artifacts of challenging cases are better dealt with L_1 norm, or their proposed combination of the L_1 norm with MS-SSIM. Meanwhile in [KCBI19] alternative metrics to PSNR are also considered.

The major difficulty in training data-driven demosaicking and denoising methods is the difficulty to obtain realistic datasets of pairs of noisy RAW and ground truth RGB images. For this reason demosaicking networks are trained with simulated data generated by mosaicking existing RGB images. However simulated data follows a statistic that can be different from real data. The RGB images used for training have already been processed by a full ISP (Image Signal Processors) pipeline which includes demosaicking and denoising steps which leave their footprint on the output image. Additionally, the Poisson noise model is only an approximation to the real noise of a specific camera. Several factors can cause deviations. For example the noise can have spatial variations due to temperature gradients in the sensor, or caused by the vignetting or the electronic components in its surroundings.

The need for a specific treatment of realistic noise has been identified in the denoising literature. Indeed most of the existing works target synthetic types of noise, e.g. Gaussian noise. Since the noise distribution is well defined, specific methods can be crafted [DF06, LBM13a, GZZF14] and data can be simulated with ground truth so to train neural networks [ZZC+17a, ZZZ18]. However, it has been shown recently in [PR17] and [ALB18] that networks trained on synthetic noise often fail to generalize to realistic types of noise. This has started a trend of study of "real noisy images". For example [CCXK18, GYZ+18] acquire datasets where a low-noise reference image is created by using a longer exposure time. Creating this type of dataset is time consuming and prone to bias, as to avoid motion blur in the long exposure the images need to be acquired with a tripod and the scene has to be static.

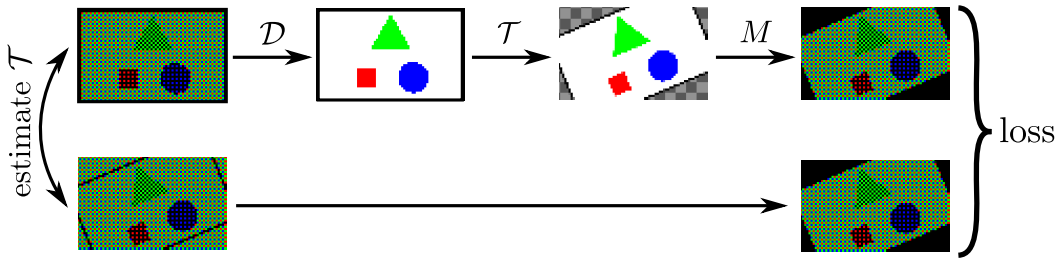


Figure 8.2: Proposed pipeline to train for demosaicking without using any ground truth. The output after applying the network $\mathcal{D}$ on the first image is warped using the transform $\mathcal{T}$ and masked with M so to be compared to the second masked mosaicked image. The black corners seen in at the last stage of the diagram indicate the undefined pixels after the transform, which are not considered by the loss.

Contribution In this chapter we introduce a mosaic-to-mosaic training strategy analog to the noise-to-noise [LMH+18] and the model-blind fine-tuning presented in Chapter 6 frameworks to be able to handle mosaicked RAW data. The trained network learns to interpolate two thirds of the image data, without having ever seen a complete image. This allows us to train both demosaicking and joint demosaicking and denoising networks without requiring ground truth. The resulting networks attain state-of-the-art results, thus eliminating the need to simulate simplistic noise models or to capture time-consuming datasets with long exposure reference frames. Although we show results only with a Bayer pattern, our method can equally be applied to other CFA patterns, such as the Fujifilm X-Trans. To the best of our knowledge, this is the first method that learns joint demosaicking and denoising without any ground truth whatsoever; the network has only seen noisy mosaicked images.

With the proposed framework, we can fine-tune a pre-trained network to a RAW burst. This allows leveraging the already available multi-frame burst data that is present on many mobile camera phones [WGDE+19]. The fine-tuning not only adapts the network to the specificities of the camera noise, but it also overfits to the burst. We demonstrate that this overfitting, when controlled, can be beneficial. A similar conclusion in the context of single-image super-resolution was reached by the authors of [SCI18]. Additionally, when used with an L_1 loss, the fine-tuned network naturally handles noise clipping, a common but challenging problem [LMH+18,ZSC19].

The proposed strategy can be used to fine-tune other demosaicking networks, for example in this chapter we show this for the network of [GCPD16] (see Figure 8.1), but it could be used in conjunction with more recent burst denoising networks such as [GMU18a] and [MBC+18b] adapted for CFA images.

The rest of the chapter is organized as follows. In Section 8.2 we present the proposed mosaic-to-mosaic training of a demosaicking network from a dataset of RAW mosaicked data without ground truth. In Section 8.3 we address the problem of joint demosaicking and denoising given a burst of RAW mosaicked noisy images. Results are shown in Section 8.4.

8.2 Learning demosaicking w/o ground truth

In this section, we propose a learning method to train demosaicking networks without any ground truth RGB images. Consider two different mosaicked pictures of a same scene I_1 and I_2. We shall use one image as partial ground truth to learn demosaicking the other (provided that there is a slight movement between the two, so that with high probability the mosaic patterns do not match).

Our method requires that the two pictures can be registered, which is possible when the viewpoints are not too different. This condition is typically met for bursts of images. Modern cameras

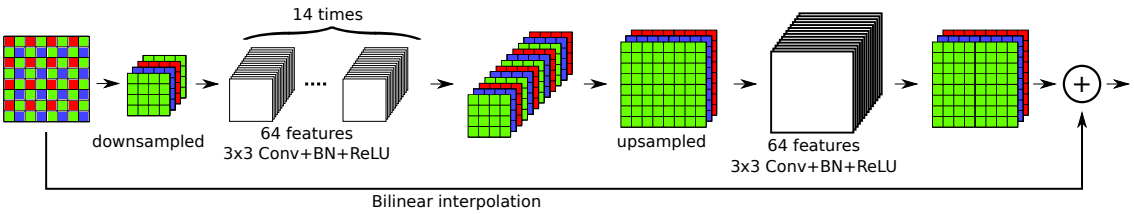


Figure 8.3: Architecture of the network used to compare the performance of learning on RGB ground truth or only with pairs of RAW images.

systematically take bursts of images, these sequences allow to eliminate shutter lag, to apply temporal noise reduction, and to increase the dynamic range of the device. Nevertheless, the pair of pictures can also be acquired manually by taking two separate pictures of the same scene.

In the following, we suppose we have a set of pairs of images (for example extracted from bursts), where each pair (I_1, I_2) consists of pictures of the same scene for which we have estimated a transformation $\mathcal{T}$ that registers I_1 to I_2. An affinity is often sufficient to accurately register a pair of images from a burst. Pairs that cannot be registered accurately won't be processed together. The original mosaicked image can be obtained from its demosaicked one by masking pixels. Thus, if we apply a demosaicking network $\mathcal{D}$ to I_1, then apply the transformation $\mathcal{T}$ followed by the mosaicking mask, we are supposed to get I_2. We can compute $M(\mathcal{T}(\mathcal{D}(I_1)))$, where M represents the mosaicking operation (masking pixels), compute a distance to I_2, which acts as ground truth, and backpropagate the gradient to train $\mathcal{D}$. In some sense, I_2 acts as a partial ground truth, as only parts of $\mathcal{T}(\mathcal{D}(I_1))$ gets compared to I_2 (for a Bayer pattern, one fourth is compared for the red and blue channels, one half for the green channel). However, contrary to artifical RGB ground truths, we do not suffer from bias introduced by the RGB processing pipeline, nor require complex settings to produce these RGB ground truths. We implemented $\mathcal{T}$ with a bicubic interpolation through which gradient can be backpropagated. This results in the following loss:

$$\ell_p(\mathcal{D}(I_1), I_2) = \|M(\mathcal{T}(\mathcal{D}(I_1))) - I_2\|_p^p, \tag{8.1}$$

where $p = 1, 2$. The norm is computed only in the pixels where both images are defined. In this section we use $p = 2$ (squared L_2 norm). The training method is depicted in Figure 8.2. Our learning process can be linked with [LMH+18, BR19]. The main difference is that we have an *a priori* on the position of the degraded pixels.

Demosaicking network To test the proposed training, we will use in this section a network architecture heavily inspired by the one from Gharbi *et al.* [GCPD16], as we have seen Chapter 7 that it performs very well, while using improvements suggested in more recent work with the usage batch normalization layers [IS15] as well as residual learning [HZRS16]. These techniques are known to speed-up training time and sometimes increase performance. The network starts with a four-channel Bayer image that goes through a series of 14 Conv+BN+ReLu layers with 64 features and 3×3 convolutions. A $15th$ layer of Conv+BN+ReLu produces 12 features with 3×3 convolutions. It is followed by an upsampling layer producing an RGB image of twice the width and twice the height. Like Gharbi et al. we added a layer (a Conv+BN+ReLu with 3×3 convolutions) before the layer producing the final output. We need add the bilinearly interpolated RGB image to the output of our network to produce the final result. All convolution layers have padding to keep the resolution constant from beginning to end. The architecture of the network is depicted in Figure 8.3.

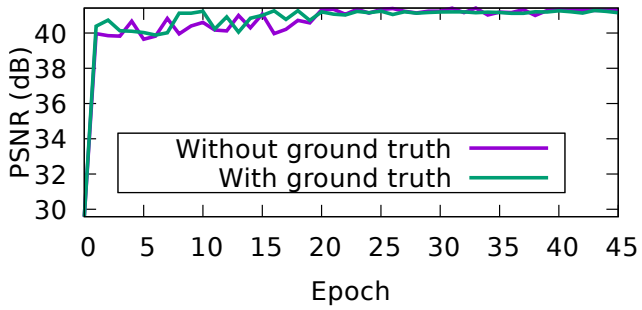


Figure 8.4: Evolution of the average PSNR on the validation dataset when training with ground truth data and when training without RGB ground truth data available. Training without RGB ground truth behaves the same than training with an RGB ground truth.

Comparing learning with ground truth RGB and our method We verify that this method for training demosaicking without ground truth is competitive with classic supervised training by training the *same* architecture with both methods and show comparable results. For this experiment we considered a mosaicking with Bayer pattern which is the most frequent mosaicking pattern.

In order to be able to compare the results of training with and without ground truth, we decided to simulate the pairs on which the demosaicking is trained. For both trainings we use the dataset of [SCC18], which consists of 500 images (of sizes around 700×500) from Flickr. To generate pairs to learn with our method, we warped the same RGB image with a random affinity - thus simulating two views - and generated the mosaicked images from them. To speed-up the training we chose the same transform for all patches of a same batch. We trained both networks for 45 epochs using Adam and a learning rate of 10^{-2}. We reduced the learning rate by a factor of 10 at epochs 20 and 40.

Figure 8.4 compares the evolution of the PSNR on the validation dataset (we use here the Kodak dataset[1]) while training our network with ground truth against the training without ground truth. It can be observed that training without ground truth behaves the same as with the ground truth. The convergence speed seems to be equivalent as well as the final demosaicking quality.

Table 8.1 shows the quality of demosaicking using either ground truth or no ground truth versus the state of the art in image demosaicking. The model learned without having ever seen an RGB image is able to achieve the same quality than the same network trained using the RGB ground truth, which indicates that having a ground truth is not necessary to obtain state-of-the-art performance on this task. For comparison, we also show the results obtained with model-based methods [Get11, HEK+14] that do not need training with ground truth (they do not need training at all).

8.3 Joint demosaicking and denoising by fine-tuning on a burst

In the previous section, we demonstrated that having a training dataset with RGB ground truth is not mandatory to reach state-of-the-art performance: Similar demosaicking performance is reached with a just a database of pairs of mosaicked data. While this was demonstrated on synthetic noise-free images, it can also be done when images are actual noisy RAW data. In this section we show an application of the method to online fine-tuning on bursts. We present the method on two networks: the noiseless network from Section 8.2 where the network has to learn

[1] http://r0k.us/graphics/kodak/

Method	With ground truth	Without ground truth
Getreuer *et al.* [Get11]	-	38.1
Heide *et al.* [HEK+14]	-	40.0
Gharbi *et al.* [GCPD16]	41.2	-
Network from §8.2	41.2	41.3

Table 8.1: PSNR results for different demosaicking methods on the Kodak dataset. Training without ground truth (network from §8.2) outperforms the methods without ground truth while still achieving state-of-the-art PSNRs.

to denoise using only the burst (this is a sort of a toy example) and the state-of-the-art network from [GCPD16].

Joint demosaicking and denoising without ground truth Using the noise-to-noise framework presented in [LMH+18], we aim to train a network with parameters θ. Supervised learning of a joint demosaicking and denoising network corresponds to solving

$$\arg\min_{\theta} \sum_i L(f_\theta(x_i), y_i), \tag{8.2}$$

where the x_i are noisy mosaicked images, and the y_i are their ideal noise-free demosaicked images, L is a loss such as L_2 or L_1. In the noise-to-noise framework, the equivalent problem (conditionally on the noise being mean preserving for L_2, or median preserving for L_1) is to solve

$$\arg\min_{\theta} \sum_i L(f_\theta(x_i), x'_i), \tag{8.3}$$

where x_i and x'_i are two noisy observations of the same scene.

Our proposal is to solve

$$\arg\min_{\theta} \sum_i \ell_1(f_\theta(x_i), x'_i), \tag{8.4}$$

where the (x_i, x'_i)s are pairs of noisy mosaicked images of the same scene and ℓ_p is the loss introduced in Equation (8.1). We use $p = 1$ in this section (L_1 norm), which allows to handle clipped noise (see discussion on the choice of the loss).

The loss requires the computation of a transform $\mathcal{T}$ matching each pair of mosaicked images. For that we use the inverse compositional algorithm [TRU98, BM01] to estimate a parametric transform (in practice we estimate an affinity which we found to be well-suited for bursts). An implementation of this method is available in [BFS18]. The advantage of this method is that it is robust to noise and can register two images very precisely (provided that they can be registered with an affinity). Since we only have access to Bayer images of size $W \times H$, the first step is to generate four-channel images of size $\frac{W}{2} \times \frac{H}{2}$ corresponding to the four phases of the Bayer pattern. The transform is then estimated on these images before upscaling it to the correct size.

Having the pairs with the associated transform, one can finally apply the pipeline presented in Section 8.2 and in Figure 8.2. As in [EDM+19] we initialize the network using a pretrained one. In particular in the following, we use the network trained for demosaicking without ground truth presented in Section 8.2, as well as the network from [GCPD16].

Choice of Loss One particularly well known problem with denoising is clipped noise: The underlying signal I belongs to a fixed range, but the noise can make it leave that intensity range. Due

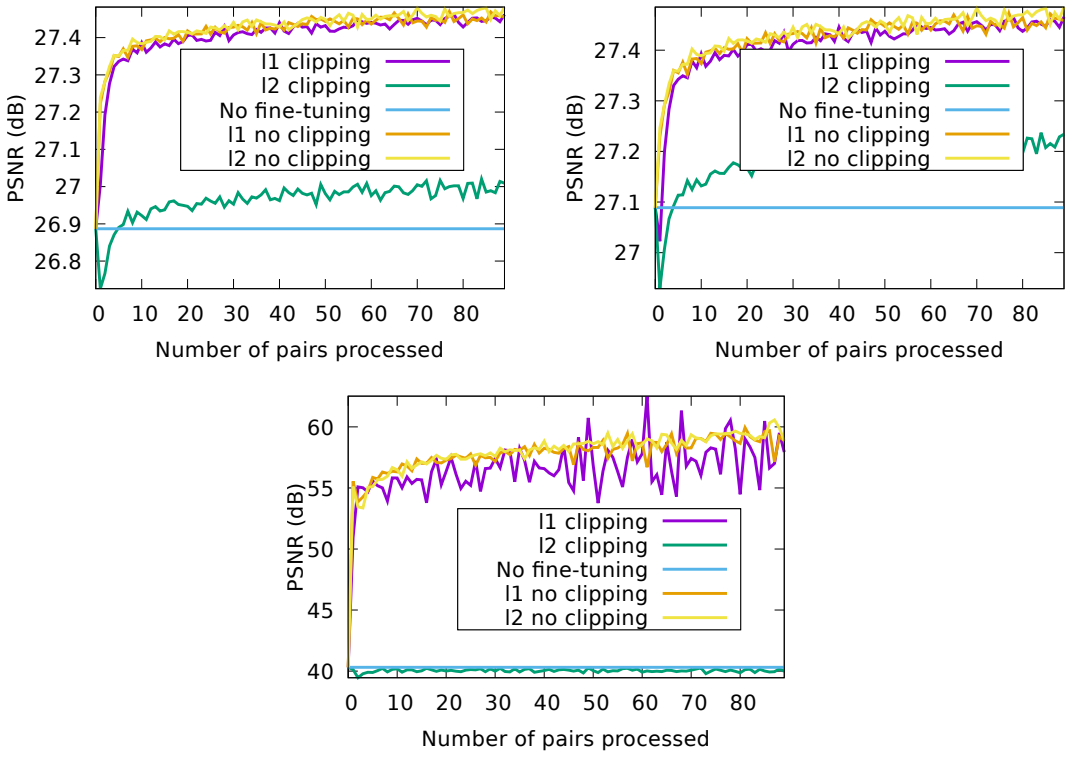


Figure 8.5: Fine-tuning a denoising network (DnCNN [ZZC+17a] $\sigma = 25$) on a burst of 10 noisy ($\sigma = 25$) grayscale images with saturated regions. From left to right: PSNRs over the whole image, on non-saturated regions, and on the saturated regions. After fine-tuning, the network works better both on saturated and non-saturated regions. While L_2 is not able to deal with clipping, using L_1 for fine-tuning performs similarly to fine-tuning without clipping.

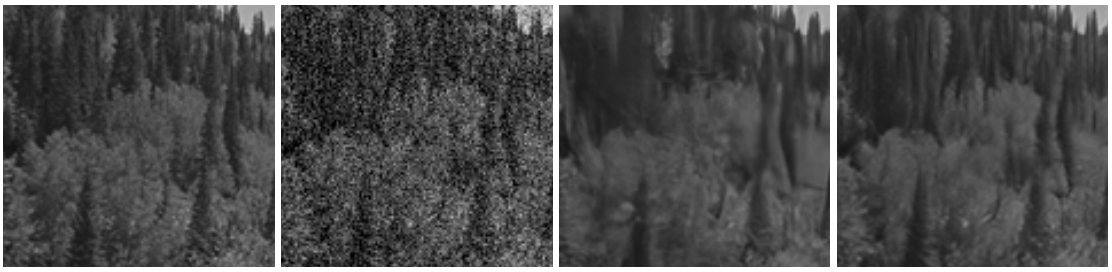


Figure 8.6: From left to right: reference image, noisy ($\sigma = 25$), pretrained DnCNN and DnCNN after fine-tuning. The details, such as the trees, are sharper and more distinguishable after fine-tuning. Figure best visualized zoomed-in on a computer.

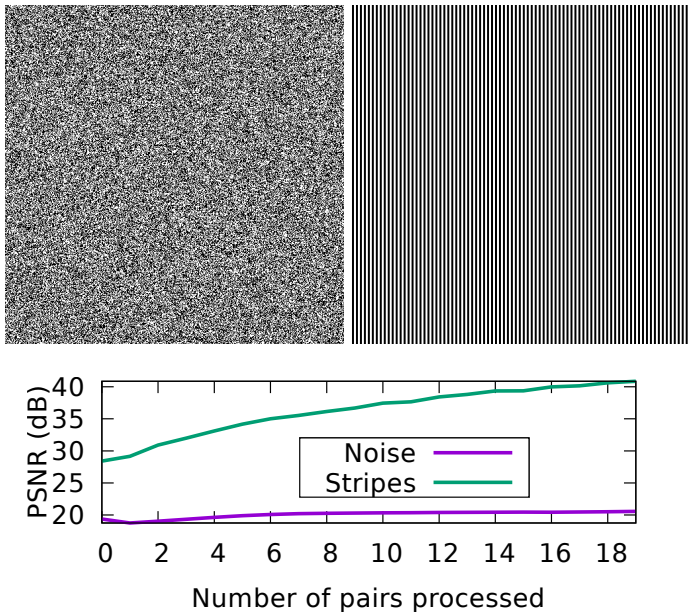


Figure 8.7: From left to right: image of binary noise and an image of stripes. Fine-tuning DnCNN on the very self-similar image of stripes leads to a much bigger increase in quality compared to the image of binary noise.

to hardware clipping, the measured image is inside the fixed range, and thus the noise statistics are biased. When minimizing with the L_1 norm over the same image with several noise realizations, the best estimator is the median of the realizations [LMH+18], which is unaffected by the hardware clipping. Thus by using L_1 norm and fine tuning on a burst, our method handles clipping without any pre or post-processing required. This phenomenon is illustrated in Figure 8.5 with a classic denoising network, DnCNN [ZZC+17a].

Fine-tuning to a single scene By fine-tuning over a single burst the network ends up overfitting the data. Usually overfitting to the training data is avoided as it results in a poor generalization ability. However, in our case the fine-tuned network will only be applied to that burst, and overfitting improves the result for that specific burst. There are other examples in the literature where a network is overfitted to a specific input (or a small dataset of inputs). For example, [CMPT+17] turns an object classification network into a video segmentation one by fine-tuning it on the first frame, which is labeled. The network then learns to track the labeled objects in the following frames.

This fine-tuning is also reminiscent of traditional image processing methods that fit a model

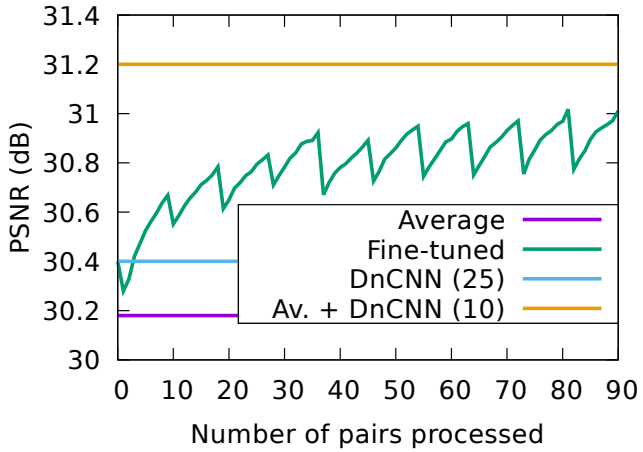


Figure 8.8: Fine-tuning a pre-trained denoising network (DnCNN $\sigma = 25$) to a specific sequence increase the quality of the result. The visible drops correspond to each change of image considered (pairs are considered in lexicographical order). It is important to finish with the reference image as to maximise the performance. The fine-tuned network, which takes as input a single frame, comes close to the performance of DnCNN applied to the temporal average of all frames.

to the patches of the image. In [YSM12] the image patches are modeled using a Gaussian mixture model (GMM), in [EA06] by representing them sparsely over a learned dictionary, and in [MSH08] via sparse convolutions over a set of kernels. In all these cases the models were trained on the input image. The assumption underlying these methods is that images are self-similar and highly redundant, allowing for compact representations of their patches.

Figure 8.6 shows that fine-tuning a grayscale denoising network (DnCNN) on a burst of images can significantly improve the denoising results. The likely explanation is that the network is able to capture a part of the image self-similarity, similar to the model-based methods. Figure 8.7 illustrates the performance evolution when fine-tuning a denoising network on a set of noisy realizations of two synthetic images, one of stripes (thus very self-similar) and a binary noise image (thus not self-similar). The performance gap is explained by the self-similarity of the former image.

8.4 Experimental results

To evaluate quantitatively the performance of the proposed training strategy, we first apply it on simulated data, since there are no real noisy raw bursts with ground truth publicly available. We generate the burst from a single image by applying random affinities. In the cases where noise is considered, the added noise is white Gaussian. During training, the affinities are estimated from the noisy raw data. Code to reproduce the results is available on `https://github.com/tehret/mosaic-to-mosaic`

Network fine-tuning on a sequence outperforms single image denoising Fine-tuning a network to a sequence allows to restore the image beyond the performance of a single image denoising. In this paragraph we consider a sequence of images with no motion and without mosaicking pattern. Figure 8.8 shows the PSNR evolution as the fine-tuning processes all the pairs of a sequence of 10 frames (90 pairs in total) without mosaicking pattern nor motion is considered. We consider the pairs in lexicographical order, that is every time a new input image is selected it is sequentially paired with all other images in the sequence. Note the characteristic shape traced by

Figure 8.9: From left to right: reference image, the network from §8.2, the network from §8.2 after fine-tuning and Gharbi *et al.*. Because of the reduced size of the training set our blind network still has some *moire* artifact but they completely disappear after fine-tuning on the data achieving a result visually close to Gharbi *et al.*without having to learn on a specific well-chosen dataset. Figure best visualized zoomed-in on a computer.

Method	kodim19	Kodak dataset
§8.2 fine-tuned on kodim19	**44.4**	40.4
§8.2 without fine-tuning	42.1	**41.3**

Table 8.2: PSNR results using an fine-tuned network on the lighthouse image of the Kodak dataset (kodim19) versus the same network without fine-tuning. While fine-tuning improves on the specific image, the overall performance on the dataset is decreased.

the PSNR curve: every time a new input image is selected the performance first drops and then steadily improves surpassing the previous peak. This shows that not only the network is adapting the current input image but it is also building upon previously seen images.

We argue that this fine tuning can be linked to a temporal noise reduction (TNR). We compare our fine-tuning based denoising strategy with three other denoising strategies: simply averaging the frames ("Average"), which amounts to a naive TNR, denoising a single frame with DnCNN ("DnCNN (25)"), which doesn't take advantage of the multiple frames, and denoising the naive TNR result with the optimal DnCNN ("Av. + DnCNN (10)"), which amounts to the best possible TNR result in this ideal case. Note that the fine tuning is largely surpassing the performance of single image denoising and approaches TNR with DnCNN. In practice temporal averaging followed by denoising is not as straightforward on mosaicked images, so there is no equivalent of this upper bound on mosaicked images. This justifies the relevance of the proposed method.

Improving demosaicking by fine-tuning Similarly to denoising, fine-tuning improves demosaicking. The evolution of the improvement, showed in Figure 8.10, is quite similar to the one presented for denoising. Moreover artifacts that existed in the initial network, due to a low amount of training, are removed completely by the fine-tuning, see Figure 8.9. The result then looks visually very similar to the result from Gharbi *et al.*that was trained specially to deal with these difficult cases.

Table 8.2 compares the PSNR obtained for different networks on the Kodak dataset. The network from Section 8.2 was fine-tuned on *kodim19*, which is singled out in the table. As expected, the fine-tuned network works well on the reference image but its performance decreases on the other images. The network without fine-tuning performs better on the whole Kodak dataset than the network that was fine-tuned on a specific image. The increase in performance for this reference image after fine-tuning was of more than $2dB$.

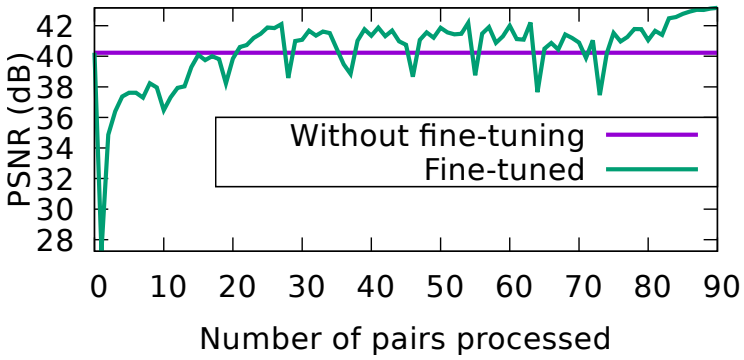


Figure 8.10: Fine-tuning a pre-trained demosaicking network (from Section 8.2) to a specific sequence increase the quality of the result. The visible drops correspond to each change of image considered (pairs are considered in lexicographical order). It is important to finish with the reference image as to maximise the performance.

Images, noise level	Methods			
	[GCPD16]	[KL18b]	§8.2 + our fine-tuning	[GCPD16] + our fine-tuning
01, $\sigma = 5$ (10 images)	34.9/.9584	34.5/.9540	35.1/.9545	**35.9/.9633**
13, $\sigma = 5$ (10 images)	32.9/.9574	32.3/.9515	33.6/.9587	**34.3/.9641**
16, $\sigma = 5$ (10 images)	37.1/.9496	36.5/.9390	36.1/.9399	**38.2/.9570**
19, $\sigma = 5$ (10 images)	36.1/.9430	35.5/.9380	36.3/.9375	**37.3/.9500**
All, $\sigma = 5$ (10 images)	36.2/.9465	35.2/.9329	36.0/.9401	**37.6/.9559**
19, $\sigma = 10$ (10 images)	33.2/.8958	31.1/.8612	32.9/.8877	**34.0/.9067**
19, $\sigma = 10$ (20 images)	33.2/.8958	31.1/.8612	33.2/.8935	**34.3/.9091**

Table 8.3: PSNR results of different methods for the task of joint demosaicking and denoising. It shows that even though our method is completely blind, it is able to compete with the state of the art. The rows identify different images from the Kodak dataset, and noise levels. Moreover increasing the length of the burst also allows to improve the quality in the cases where it might perform worse otherwise. Our method used the network trained in Section 8.2 and was fine-tuned with 10 generated noisy images except when mentioned otherwise.

Joint demosaicking and denoising using fine-tuning The final application of fine-tuning is to do both previous applications at the same time. Table 8.3 compares our method to two other methods of joint demosaicking and denoising. The networks were fine-tuned on each image individually. Overall our fine-tuning approach is very competitive. A network that has not been trained with noisy data prior to fine-tuning (§8.2 + our fine-tuning) is now able to perform at the same level as one of the best network trained for this specific application. When using the state-of-the-art network from [GCPD16], our fine-tuning improves the final quality by more than $1dB$.

Not only do we achieve competitive results in terms of PSNR, the results are free of demosaicking artifacts. Indeed, as shown in Figure 8.1, even in the regions that are particularly hard such as the fence. For example there is no *zippering* compared to the result of Gharbi *et al.*.

The final experiment is on real data. We took a burst from the HDR+ dataset [HSG+16] and applied our process. We compare the result of a simple bilinear interpolation, the result of [GCPD16] and [GCPD16] with our fine-tuning in 8.11. Fine-tuning allows for a better denoising

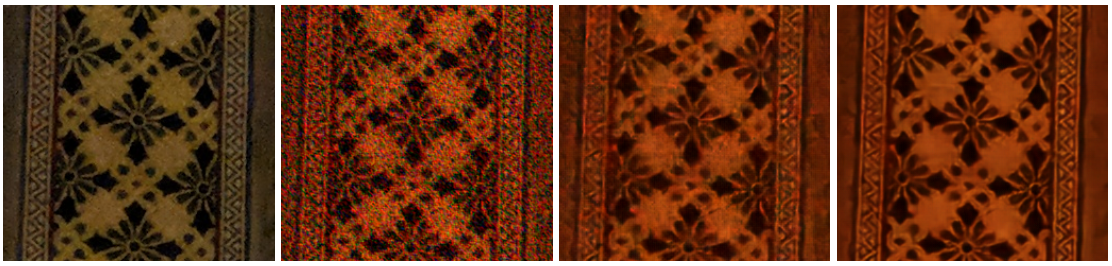


Figure 8.11: Experiment on a real burst. Left to right: The result of the HDR+ pipeline [HSG+16], bilinear interpolation, [GCPD16] and [GCPD16] with our fine-tuning. Contrast was enhanced for all methods except HDR+. Note that the HDR+ pipeline includes color balance as well as sharpening. It also uses all the images of the burst to produce the result (all other methods use only the reference frame). Figure best visualized zoomed in on a computer.

and a better reconstruction of details while limiting artifacts. We present additional result on the HDR+ dataset in Figures 8.12, 8.13, 8.14. In these figures we show the result from the HDR+ pipeline [HSG+16], a bilinear interpolation, the result of Gharbi *et al.* [GCPD16], and the result of Gharbi *et al.* [GCPD16] with our fine-tuning.

Remarks on computation cost We empirically found that the amount of data needed for fine-tuning the network is linked to the number of pixels and not the number of images of a burst. This allows to fine-tune even on short bursts like the ones from the HDR+ dataset (of size 2400×1300) using at most six images. Regarding computation time, we presented fine-tuning as an offline application, for example for professional photography where best quality is required. However, recent works [TRJ+19, TTP+19] have shown that fine-tuning can also be achieved in real time for videos.

8.5 Conclusion

In this chapter, we have proposed a novel way of training demosaicking neural network without any RGB ground truth, by using instead other mosaicked data of the same scene (such as from a burst of images). Based on it and on recent neural network advances, we proposed a method to train jointly demosaicking and denoising with bursts of noisy raw images. We show that fine-tuning on a given burst both boosts the reconstruction performance and handles clipped noise natively. It also presents a specific case where overfitting a network to the training data is valuable. Since we do not expect generalization there's only benefits from this overfitting.

We hope our work can lead to new camera pipeline calibration procedures, and general improvement of the image quality when a burst is available.

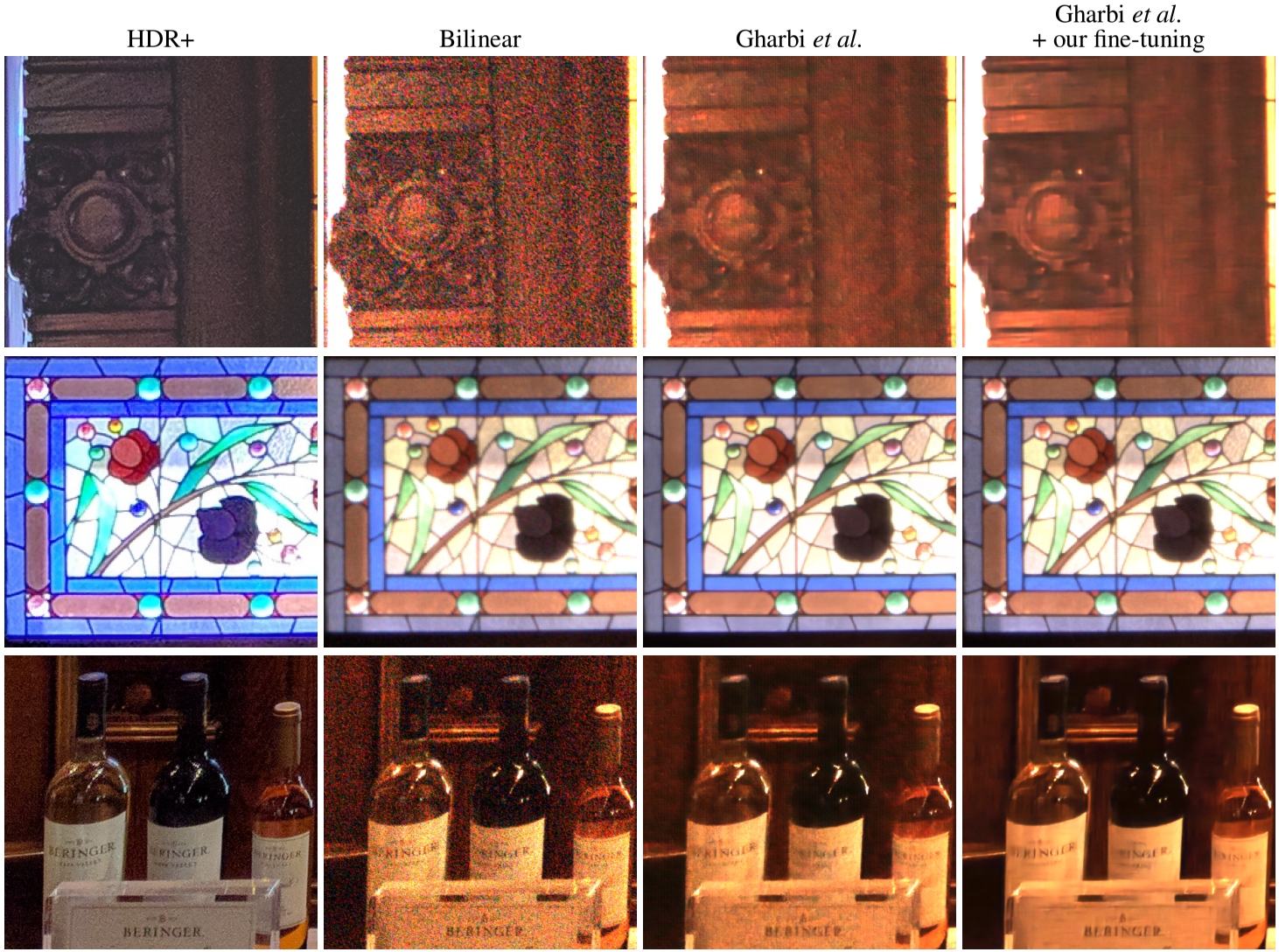


Figure 8.12: Experiment on a real burst. Left to right: The result of the HDR+ pipeline [HSG+16], bilinear interpolation, Gharbi *et al.* [GCPD16], and Gharbi *et al.* [GCPD16] with our fine-tuning. Contrast in dark regions was enhanced for all methods. Note that the HDR+ pipeline includes color balance as well as sharpening. It also uses all the images of the burst to produce the result (all other methods use only the reference frame). Figure best visualized zoomed in on a computer.

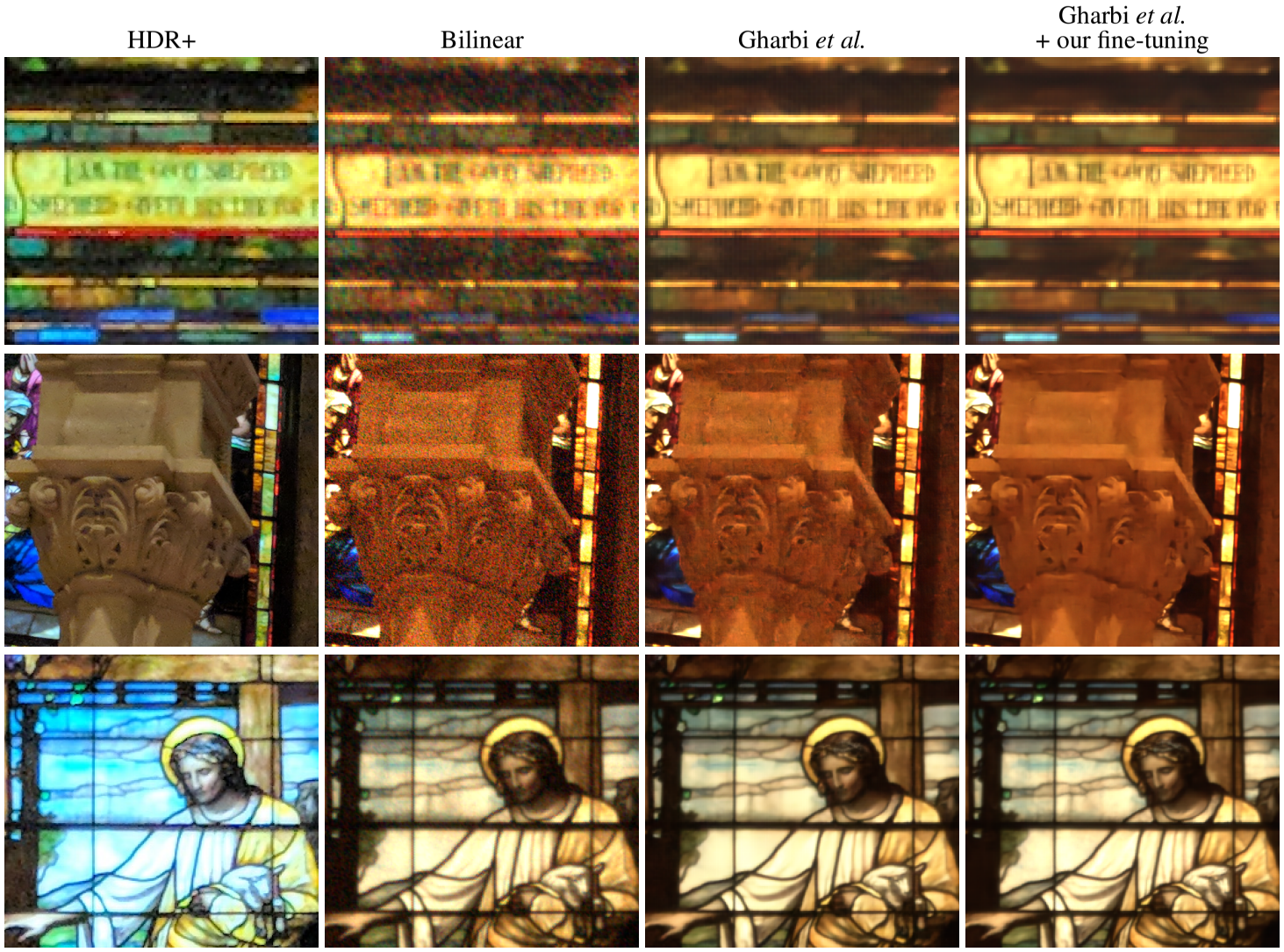


Figure 8.13: Experiment on a real burst. Left to right: The result of the HDR+ pipeline [HSG+16], bilinear interpolation, Gharbi *et al.* [GCPD16], and Gharbi *et al.* [GCPD16] with our fine-tuning. Contrast was enhanced for all methods except HDR+. Note that the HDR+ pipeline includes color balance as well as sharpening. It also uses all the images of the burst to produce the result (all other methods use only the reference frame). Figure best visualized zoomed in on a computer.

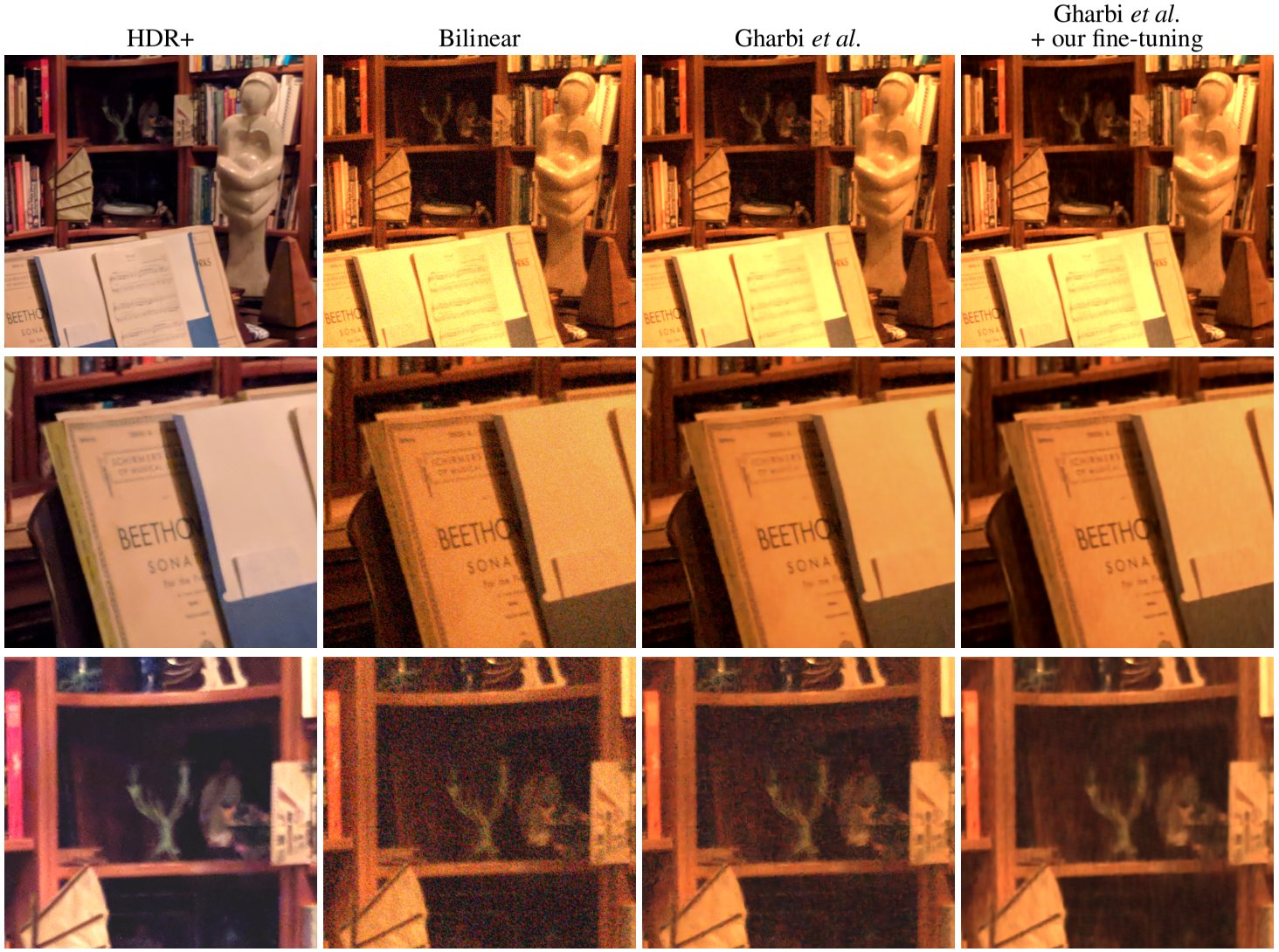


Figure 8.14: Experiment on a real burst. Left to right: The result of the HDR+ pipeline [HSG+16], bilinear interpolation, Gharbi *et al.* [GCPD16], and Gharbi *et al.* [GCPD16] with our fine-tuning. Contrast in dark regions was enhanced for all methods. Note that the HDR+ pipeline includes color balance as well as sharpening. It also uses all the images of the burst to produce the result (all other methods use only the reference frame). Figure best visualized zoomed in on a computer.

Part III

Application of patch-search to detection

9 How to Reduce Anomaly Detection in Images to Anomaly Detection in Noise

Anomaly detectors address the difficult problem of detecting automatically exceptions in a background image. The background image can be as diverse as representing fabric or a mammography. Detection methods have been proposed by the thousands because each problem requires a different background model. By analyzing the existing approaches, we show that the problem can be reduced to detecting anomalies in residual images (extracted from the target image) in which noise and anomalies prevail. Hence, the general and impossible background modeling problem is replaced by a simple noise model, and allows the calculation of rigorous detection thresholds. Our approach is therefore unsupervised and works on arbitrary images. The residual images can favorably be computed on dense features of neural networks. Our detector is powered by the a contrario detection theory, which avoids overdetection by fixing detection thresholds taking into account the multiple tests. This work was published in [DEMD18] and [EDMD19].

9.1 Introduction

The automatic detection of anomalous structure in arbitrary images is concerned with the problem of delineating image regions not conforming with the rest of the image. This is a challenging computer vision problem, as there seems to be no straightforward definition of what is (ab)normal for a given image.

Anomalous structure in images can be generally described as being either caused by high-level or low-level outliers. High-level anomalies are related to the semantic information presented in the scene. For example, human observers immediately detect a person inappropriately dressed for a given social event. In this chapter, we focus on the problem of detecting anomalies due to low- or mid-level rare events (e.g., patterns) present in images. This is an important problem in many industrial or biomedical applications where a fast and reliable way of detecting rare patterns is needed.

We propose here an unsupervised method for detecting anomalies in an arbitrary image. The method doesn't rely on a training dataset of normal or abnormal images, neither on any other prior knowledge about the image statistics. It directly detects anomalies with respect to residual images estimated solely from the image itself. We only use a generic, qualitative background image model: we assume that anything that repeats in an image is *not* an anomaly. Hence we

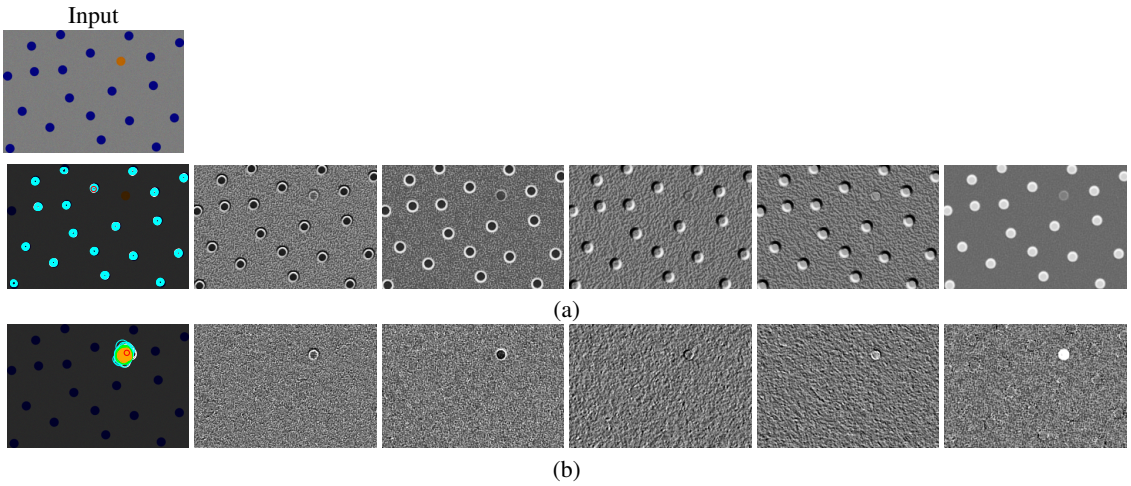


Figure 9.1: Image anomalies are successfully detected by removing all self-similar content and then looking for structure in the residual noise, which parameters are easy to estimate. First row: an image with a color anomaly (the red dot); (a): the detections (first column) obtained from principal components of CNN feature maps (2nd-5th columns) (b): the detection map on the same features obtained after removing the self-similar content. In the detection maps, cyan means good detection and orange extremely salient detection.

extract from the input image a residual image containing everything that does not repeat. This unstructured image is akin to noise, but still contains the anomalies if any.

Detecting anomalies in noise is far easier and can be made rigorous and unsupervised by the *a-contrario* theory [DMM07] which is a probabilistic formalization of the *non-accidentalness* principle [Low85]. The *a-contrario* framework has produced impressive results in many different detection or estimation computer vision tasks, such as, segment detection [VGJMR10], ellipse detection [PGvG12], spots detection [GM09], vanishing points detection [LGvGRM14], fundamental matrix estimation [MS04], mirror-symmetry detection [PGvGO13], among others. The fundamental property of the *a-contrario* theory is that it provides a way for automatically computing detection thresholds that yield a control on the number of false alarms (NFA). It follows that not only one can detect anomalies in arbitrary images without complex modeling, but in addition the anomalies are associated an NFA which is often very small and therefore offers a strong guarantee of detection.

In a nutshell, our method removes from the image its self-similar content (considered as being normal). The residual can be modeled after a simple equalization as Gaussian noise, but still contains the anomalies according to their definition: they do not repeat. We shall show detections performed directly on the residual, or alternatively on residuals extracted from dense low- and mid-level features of VGG [SZ15] the popular pre-trained deep neural network. Our method is general and works equally well on these different image representations.

A preliminary short version of this work was published in a conference [DEMD18]. The anomaly detection method developed here is also described briefly in our review paper [EDMD19]. Our Section 2 below summarized some of the conclusions about the literature contained in this last paper. The present version incorporates a more detailed analysis of the detection method and its implementation.

The remainder of the chapter is organized as follows. Section 9.2 discusses in detail previous work and the substantial differences to what we propose. Section 9.3 explains the method and its implementation. In Section 9.4 we present results of the proposed method on both real and synthetic data, comparing the algorithm to other state-of-the-art anomaly detection methods. We finally close in Section 9.5.

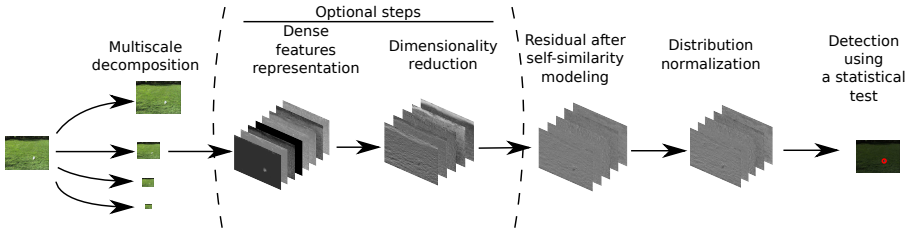


Figure 9.2: Our method applies the same framework at all the different scales. The framework computes the difference between the image (possibly using a dense representation using a neural network) and its self-similar model before normalizing this difference output distribution. The detection is then done on these normalized distributions. The corresponding pseudocode is presented in 20.

9.2 An analysis of the literature

The 2009 review [CBK09] examining some 400 papers on anomaly detection considered allegedly all existing techniques and all application fields. It is fairly well completed by the more recent [PCCT14] review. These reviews agree that classification techniques like SVM can be discarded, because anomalies are generally not observed in sufficient number and lack statistical coherence. There are exceptions like the recent method [DLBM14] which defines anomalies as exceptional events that cannot be learned, but after estimating a background density model, the right detection thresholds are nevertheless learned from anomalies. A broad-related literature exists on saliency measures, for which learning from average fixation maps by humans is possible. For example [TRH11] trained on average human fixation maps to learn both the anomalies and their surround vectors as Gaussian vectors. This reduces the problem to a two class Bayesian classification problem. The goal of saliency detectors is only to deliver a fuzzy saliency map. Anomaly detectors instead signal anomalous regions. The saliency detectors try to mimic the human visual perception and in general introduce semantic prior knowledge related to the perceptual system (e.g., face detectors). This approach works particularly well with neural networks because attention maps obtained by gaze trackers can be used as a ground truth for the training step. SALICON [HSBZ15] from Huang *et al.*is one of these deep neural networks architecture achieving state of the art performance. In [Kum03] a neural network is trained on a base of defect/non-defect, thus again performing two classes classification. But the anomaly detection problem has been generally handled as a "one class" learning problem. The 2003 very complete review by Markou and Singh [MS03] concluded that most research on anomaly detection was driven by modeling background data distributions, to estimate the probability that test data do not belong to such distributions. Hence the mainstream methods can be classified by their approach to background modeling.

9.2.1 Probabilistic background models

Their principle is that anomalies occur in the low probability regions of the background model. This stochastic model can be parametric (Gaussian, Gaussian mixture, regression), or nonparametric. For example in "spectral anomaly detection", an anomaly is defined by having deviant coordinates with respect to normal PCA coordinates. In [AT10] a Gaussian background Fourier model of the image phase is followed by a Mahalanobis threshold. In [DZ11] a Gaussian background model from random pixels is similarly followed by a Mahalanobis threshold. In [GC04] the background is characterized in a feature space of principal components, and hypothesis testing is used for the detection of anomalous pixels. In [THCB95] the assumption is made that abnormalities are uniformly distributed outside the boundaries of normality, defined as the probability density estimation of the training data. In [HN01] Honda and Nayar introduced a generic method

which works on all type of images. The main idea is to estimate the probability of a region conditioned on the surroundings. The method employs independent component analysis to find a compact representation of the region space and its surroundings. The [GM09] method models the background image as a Gaussian stationary process. This is rather restrictive, but this precise model allows computing an accurate Number of False Alarms [DMM07] for anomalies.

Thus, in methods relying on a probabilistic background model, outliers are detected as incoherent with a probability distribution estimated from the input image(s). The anomaly threshold is a statistical likelihood test on the learned background model. We now pass to non-stochastic background models.

NN-based background reconstruction. "Replicator" neural networks [HHWB02] can model a background. These are multi-layer feed forward neural networks with same number of input and output neurons. The training involves compressing data into hidden layers. The testing phase reconstructs each data sample. Its reconstruction error for the test instance is used as an anomaly score. The work in [SSW+17] is also equivalent to using an autoencoder and looking at the norm between the original and the output. A GAN is trained (generator + discriminator) via gradient descent, a representation in latent space is computed, and the output is compared to the input. The discriminator cost is then used alongside the representation on the input by the network to find the anomalies. This paper is related to [An16] that computes a reconstruction probability from a variational autoencoder.

Fourier Background subtraction. Perhaps the most successful background based method is the detection of anomalies in periodic patterns of textile [TH99, TH03, PCC10]. This can be done naturally by cutting specific frequencies in the Fourier domain and thresholding the residual to find the defects.

9.2.2 Non constructive background models

These methods present the big advantage that they no longer require the construction of a background model, which for most images is anyway impossible. Hence, they simply make structural assumptions on the background image that would be violated by anomalies.

Center-surround enhancement. These methods are mainly used for creating saliency maps. Their rationale is that anomalies pop up as local events contrasted with their surroundings. In [IKN98], center surround detectors based on color, orientation and intensity filters are combined to produce a final saliency map. Detection is then done on a simple winner-takes-all scheme on the maximum of the response maps. In [MVOP11] a saliency map is obtained from center-surround contrast coefficients for wavelet filters. An image wavelet pyramid reconstruction with these coefficients enhances local anomalies. Detection in images and videos is also done in [GMV08] with center-surround saliency detectors which stem from [IK00] adopting similar image features. In [HN01], the main idea of a fast and general anomaly detection method is to estimate the probability of a region conditioned on the surroundings. The method employs ICA and KLT to find a compact (with elements as independent as possible) of the region space and its surroundings.

The sparsity model. A more recent nonparametric trend is to learn a sparse dictionary representing the background (i.e., *normality*) and to characterize outliers by their non-sparsity. In [MTZM13] the patch background model is simply its PCA and the patch saliency is computed as the L_1 norm of the patch coefficients in PCA. Aggregating these values gives a pixel

saliency. The saliency formation in [XM07] builds a Gaussian mixture model for patches, and probability thresholds are learned on images without anomalies. The final result is a saliency map. In [BCW14] the background model is a learned patch dictionary from a database of anomaly-free data. The abnormality of a patch is measured as its Mahalanobis distance to a 2D Gaussian learned on the parameter pairs composed by the L^1 norm of the coefficients and of their reconstruction error. An extension [CBFW15] learns a convolutional sparse dictionary. Similarly in [ESV12] a patch is anomalous when the L^1 norm of its sparse decomposition on a learned dictionary is too large.

Defining anomaly detection as a variational problem where anomalies are detected as non-sparse is also the core of the method proposed in [AEHOR15]. The L^1 norm of the coefficients on the learned background dictionary is used as an anomaly measure.

The self-similarity model. The self-similarity principle has been successfully used in many different applications. In particular in image denoising such as the bilateral filter [TM98] or non-local means [BCM05a] for example; but also for texture synthesis in the pioneering work by Efros and Leung [EL99]. The basic assumption of this generic background model, applicable to most images, is that in normal data, features are densely clustered. Anomalies instead occur far from their closest neighbors. This idea can be implemented by clustering (anomalies being detected as far away from the centroid of their own cluster), or by nearest neighbor search (NN). NN search leads to simple direct rarity measurements. For example in [SM09] the saliency measure is $S_i = \left(\sum_{j=1}^{N} \exp\left(\frac{-1+\rho(F_i,F_j)}{\sigma^2}\right)\right)^{-1}$ where F_i are local features, and F_j the closest features to F_i. If all F_j are far away from F_i, the saliency is high. The algorithm in [ZC10] is inspired from NL-means [BCM05a]: a) fix a similarity threshold learned in a reference image without anomalies and b) compare each patch of the source image to the patches of the reference; if the distance is higher than the similarity threshold, then the patch is an anomaly. A similar idea can be found back in [TD98] : "The distance of the new object and its nearest neighbor in the training set is found and the distance of this nearest neighbor and its nearest neighbor in the training set is also found. The quotient between the first and the second distance is taken as indication of the novelty of the object." The self-similarity measurement in [GZMT12] finds for each patch p_i its 64 most similar patches q_k in a spatial neighborhood and computes its saliency as $S_i = 1 - \exp\left(-\frac{1}{64}\sum_{k=1}^{64} d(p_i, q_k)\right)$.

A similar method and saliency detector is proposed in [MC14]. This method performs a dimension reduction thanks to a low dimensional embedding of a nearest neighbor graph using the coordinates of points on the eigenvectors of the graph Laplacian. The anomaly score is then given by the distance to the first nearest neighbors with the diffusion distances. In [BI07], image regions are matched (with deformation allowed) to others in the same image or video. The probability of the deformation is estimated and gives a saliency map.

Boracchi and Roveri [BR14] proposed to detect structural changes in time-series by exploiting the self-similarity. Their general idea is that a normal patch should have at least one very similar patch along the sequence. Given a temporal patch (a small temporal window) the residual with respect to the most similar patch in the sequence is computed. This leads to a new residual sequence (i.e., change indicator sequence). The final step is to apply a traditional change detector test (CDT) on the residual sequence. CDTs are statistical tests to detect structural changes in sequences, that is, when the monitored data no longer conform to the independent and identical distributed initial model. CDTs run in an online and sequential fashion.

Global rarity measurement. Generally histogram based, these methods assign an anomaly score to each tested feature based on the inverse of the height of the bin to which it belongs.

Similarly in [RMD+13] a saliency map is obtained by summing up the rarity of 32 multiscale oriented features, computed for each pixel as a weight inversely proportional to its rarity in the wavelet histogram. Patches are represented in [BI12] by their coefficients on a patch dictionary learned on natural images. The final saliency is the inverse of a patch probability of happening. The method is finally combined with a local center-surround saliency measure.

9.2.3 Discussion and our proposition

We now discuss briefly the previous classification. Background probabilistic modeling is very powerful when images belong to a restricted class of homogeneous objects, like textiles. Indeed, it furnishes rigorous detection thresholds based on the estimated probability density function. But, regrettably, this method is nearly impossible to apply on generic images. For the same reason, background reconstruction models based on CNNs are restrictive and do not rely on provable detection thresholds. Center-surround contrast methods are successful for saliency enhancement, but lack a detection mechanism. Hence, they only furnish a saliency image, not a binary anomaly decision. The sparsity and the self-similarity models are tempting and thriving. Their big advantage is their universality: they can be applied to most images. But again, excepting [BCW14], they lack a rigorous detection mechanism, because they work on a feature space that is not easily modeled.

Our proposition is to benefit of the advances of the above methods while avoiding their mentioned limitations. To this aim, we construct a probabilistic background model on a new feature image that we call the residual. This residual is obtained by reconstructing a self-similar version of the target image. The difference between the target and its self-similar version is called the *residual* and becomes our new background. Being not self-similar, this background is akin to a colored noise. Hence hypothesis testing can be applied to it, and more precisely multiple hypothesis testing (also called *a contrario* method), as proposed in [GM09].

In that way, a general and simple method can be built that works on all images and detects anomalies by a rigorous threshold. It does not require learning, and it is easily made multiscale. Our underlying model can therefore be considered as fully generic in the sense that it decomposes any image into a self-similar part and its "residual", which contains only noise and the anomalies.

9.3 Method

The anomaly detection method that we propose is therefore built on two main blocks: a removal of the self-similar part of the image, and a simple statistical detection test based on the *a contrario* framework on the residual. The pipeline is summarized in Figure 9.2 and in Algorithm 20.

9.3.1 Self-similarity Background Modeling

The proposed self-similarity based background subtraction is inspired from patch-based denoising algorithms, but with a crucial difference. All self-similarity based denoising algorithms share a similar procedure. First, a set of similar patches is computed. This search is generally performed locally around each patch [DFKE07b, BCM05a] to keep computational cost low and to avoid noise overfitting. An "self-similar estimate" of the query patch is then built using the nearest neighbors found at the previous step. These nearest neighbors will be picked with a different rule for anomaly detection than for denoising, as we shall see next.

Self-similarity from *exclusively* non-local patches. Image denoising tries to maximize the quality of the result and must therefore perform acceptably even with the non self-similar parts of

Algorithm 20: Anomaly detection algorithm

input : Input image u, reference image v, size of patch sp, number of nearest neighbors n, h the patch similarity parameter, a list R of disk radii to test, the NFA threshold ε, a neural network nn (optional)

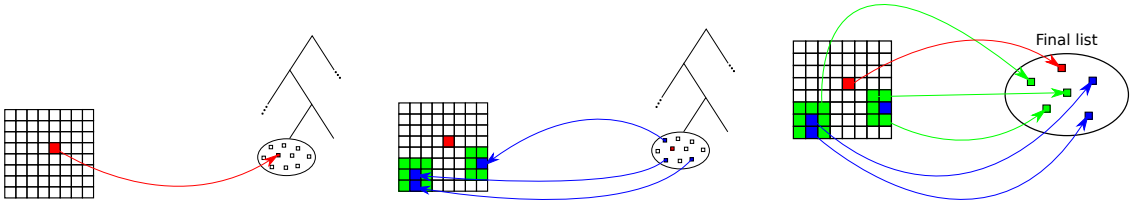
output : List of detections l (contains size and position)

$(u_i) \leftarrow multiscaleDecomposition(u, nscales)$ // See Section 9.3.2
$(v_i) \leftarrow multiscaleDecomposition(v, nscales)$ // and Algorithm 26
for *scale* $i = 0$ *to* 3 **do**
    if nn *is provided* **then**
        $\hat{u}_i \leftarrow extractFeatures(u_i, nn)$ // Use neural network features to represent u_i
        $\hat{v}_v \leftarrow extractFeatures(v_v, nn)$ // See Section 9.3.3
    else
        $\hat{u}_i \leftarrow u_i$
        $\hat{v}_i \leftarrow v_i$
    $r_i \leftarrow computeResidual(\hat{u}_i, \hat{v}_i, sp, n)$ // See Algo. 24
    $r_i \leftarrow fitResidual(r_i)$ // See Algo. 7
    $l_i \leftarrow detect(r_i, height(\hat{u}_0), width(\hat{u}_0), channels(\hat{u}_0), R, \varepsilon)$ // See algo. 27
    Add l_i to l

the image. For anomaly detection instead, the algorithm should perform well only if a clear structure emerges. Furthermore, we want our self-similarity search to be made on the whole image. But the main difference with classical denoising is that we *forbid* local comparisons. Thus, contrary to classic denoising algorithms, the search is performed *outside a square region surrounding each query patch* only. Otherwise any anomaly with some internal structure might be considered a structure. What matters is that the event represented by the anomaly is unique in the image.

Searching for similar patches. In image patch-based methods, searching for similar patches is usually done in a small squared region surrounding the query patch. For denoising, searching in a small region acts as a regularizer and speeds up the processing. In our case this can't be done because we *forbid* a local squared region around the query patch. We'd also like to take advantage of the whole image; repetitions could appear anywhere in the image and not necessarily locally. This is why we used the global search presented in Chapter 2. The search is summarized in Algorithm 23 with each step shown in Figure 9.3. Another advantage of this search is that the query image and the reference image can be different without any loss in performance. This is similar to an external search in a database made of one or multiple images (See Figure 9.4).

In this chapter we suggest using VP-trees for the global search since it has been shown to work well for a similar problem [EAM17]. However, other global patch search methods, such as PatchMatch [BSFG09a], can also be used. While PatchMatch has more convergence guarantees, see Chapter 10, it is also less computation efficient. PatchMatch is very efficient when looking for very few (namely one or two) nearest neighbors but struggles for a large number of nearest neighbors. PatchMatch also requires the computation of similar patches for the entire database while the current approach only does it for the image that is being processed. Moreover this approach is easier to parallelize than PatchMatch.



(a) Step 1: the search of the bin of the query patch in the VP-tree

(b) Step 2: the local search based on the elements found with the VP-tree

(c) Step 3: combination of the results from each local search

Figure 9.3: Steps of the VPLR search

Algorithm 21: Construction of a VP-tree

input : A set of points in a metric space $\mathcal{S}$, b the number of elements to be used for the computation of the vantage point, m the maximum size of the bins

output : The VP-tree $\mathcal{T}_\mathcal{S}$

if $|\mathcal{S}| \leq m$ **then**
    No splitting is needed
    return an empty tree
$s \leftarrow chooseVantage(\mathcal{S}, b)$ `// See Algo. 22`
Compute the median distance p from s to the other elements of $\mathcal{S}$
Construct $\mathcal{T}_1$ using the elements of $\mathcal{S}$ which are closer than p from s
Construct $\mathcal{T}_2$ using the elements of $\mathcal{S}$ which are further than p from s
return ((s,p), $\mathcal{T}_1$, $\mathcal{T}_2$)

Background model computation. The background removal step follows the steps of the Non-local means algorithm [BCM05a]. For a similarity parameter and for each patch P in the image the n most similar patches denoted by P_i are searched and averaged to get a self-similar estimate $\hat{P}$ for the patch,

$$\hat{P} = \frac{1}{\sum_{i=1}^{n} \exp\left(-\frac{\|P-P_i\|_2^2}{h^2}\right)} \sum_{i=1}^{n} \exp\left(-\frac{\|P-P_i\|_2^2}{h^2}\right) P_i. \tag{9.1}$$

Aggregating these estimated patches produce a self-similar model of the tested image. The residual is then the difference between this self-similar model and the input image. As it can be seen in Figure 9.1, this process separates the content of the image (background) from the noise and anomalies. We argue this residual is much easier to model since it doesn't contain any complex structure. Algorithm 24 gives a generic pseudocode for this process.

Residual distribution. The detection process, as presented in 9.3.2, relies on the residual being Gaussian. However, the distribution of $r(u)$ is not necessarily Gaussian. In [ZLZ+17] it is hinted that $r(u)$ might follow a Laplace distribution for natural images for some denoising algorithm. The fact that non-local means transforms white Gaussian noise into white Gaussian noise is substantiated in [BCM08]. Based on these remarks, we add an additional step to transform the residual (which can be either Laplacian or Gaussian) into the required distribution: We fit a few distributions on the residual (See Algorithm 7). The distribution depends on the image and on the choice of the input features, depending on whether we work directly on an image or on features from a neural network (as mentioned in Section 9.3.3). In the case of a neural network, Zeros are ignored when estimating the distribution. Indeed, when using CNNs, RELUs cause zero to be

Algorithm 22: $chooseVantage$: Choose a vantage point

input : A set of points in a metric space $\mathcal{S}$, b the number of elements to be used for the computation of the vantage point

output : A vantage point s

Draw a set of b points P from $\mathcal{S}$

for $s \in P$ **do**

  Compute the median distance p from s to the elements of $\mathcal{S}$

  Compute the second moment of $\{\|s - u\| - p \mid u \in \mathcal{S}\}$

return $s \in P$ with the largest second moment

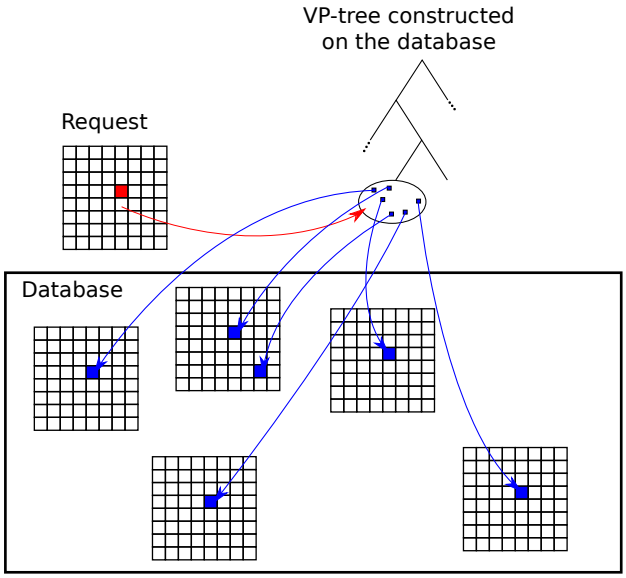


Figure 9.4: Using the global search to search in a database

over-represented. To equalize the residuals and simplify the detection model, we use a non-linear transform and rescale $r(u)$ in order to fit a centered Gaussian distribution with unit variance.

9.3.2 Statistical detection by the *a contrario* approach

The *a contrario* model. Our goal is to detect structure in the feature image residual $r(u) = \hat{u} - u$. We are in a much better situation modeling $r(u)$ than u. Indeed, contrarily to u, $r(u)$ is by construction unstructured and akin to a colored noise. We treat it as a stationary probabilistic spatial process and follow [GM09], who proposed automatic detection thresholds in any colored Gaussian noise.

Their *a contrario* framework computes a *Number of False Alarms (NFA)* for each value in excess in a colored noise image, under the null hypothesis that the residual is centered Gaussian noise of unit variance. The null hypothesis doesn't require the noise to be uncorrelated. Since anomalies are expected to deviate from this background model, this amounts to checking the tails of the Gaussian and to retain high values as significant if their tail has a very small area. Computing an NFA is justified by the intensive multiple testing involved in the detection. Indeed every pixel of each residual channel are tested.

More precisely, given a set of random variables $(X_i)_{i \in [|1,N|]}$, understood as a set of threshold tests, a function f verifies the NFA property if it guarantees a bound on the expectation of its

Algorithm 23: VPLR search algorithm

input : v an image, $\mathcal{F}$ a VP-tree forest constructed with the patches from v, p a query patch, $\kappa \times \kappa$ the size of local search regions, k the number of patches required, a forbidden region (possibly empty)

output : A list of matches for p

Initialize an empty list l
for *each tree* $\mathcal{T}$ *in* $\mathcal{F}$ **do**
    Initialize t to $\mathcal{T}$
    while t *is not a bin* **do**
        $(s, d), \mathcal{T}_1, \mathcal{T}_2 \leftarrow t$
        if *the distance from* p *to* s *is smaller than* d **then**
            $t \leftarrow \mathcal{T}_1$
        else
            $t \leftarrow \mathcal{T}_2$
    Retrieve the list $\{\varphi_1, \ldots, \varphi_k\}$ of k best matches from t that are outside the forbidden region and add them to l
Keep the k best matches from l
for *each* φ_i *in* l **do**
    Search in the region of size $\kappa \times \kappa$ centered on φ_i in v for better matches (consider only patches outside the forbidden region)
    Keep the k best matches in l
return the list $l = \{\varphi'_1, \ldots, \varphi'_k\}$ of k best matches after the update using the VPLR search

Algorithm 24: $computeResidual$: Computation of the unstructured residual

input : A multichannel image u, a multichannel reference image v, n the number of nearest neighbors, h the similarity parameter, patch size sp `// v is potentially equal to u`

output : Residual $r(u) = \hat{u} - u$ where $\hat{u}$ is the model of u.

for *all multichannel patch* P *of* u **do**
    Estimate the n nearest neighbors $(P_1, \ldots, P_n)$ of P in v **deprived from a square region around** P if v is u. `// See Algo. 23`
    Reconstruct the patch (using (9.1))
for *pixels* j *in* u **do**
    $\hat{u}(j) = \frac{\sum_{i \in \{s | j \in W_s\}} \hat{P}_i(j)}{\#\{s | j \in W_s\}}$ `// Aggregate the different estimates`
return $r(u) = \hat{u} - u$
Notation convention: W_s is the set of pixels in the patch centered at s. $\hat{P}_i(j)$ is the value at pixel j of the reconstructed patch centered at i.

Algorithm 25: $fitResidual$: Fit the residual distribution and convert it to a Gaussian

input : r a set of residual values obtained from Alg. 24
output : r' the resulting modified residual

Ignore all zeros for the distribution estimation.
for *all* α *in* $[0.5, 0.6, \ldots, 1.4]$ **do**
    Test if r^α is Laplacian
    Test if r^α is Gaussian
The best distribution D and α are kept.
// Best distribution in terms of minimizing the l^2 distance from the uniform distribution of the cumulative histogram of $cdf_D(r^\alpha)$, where D is the tested distribution.
$r' \leftarrow inv\_cdf_{Gaussian(0,1)}(cdf_D(r^\alpha))$ // Applied element-wise

number of false alarms under the null-hypothesis, namely:

$$\forall \epsilon > 0, \mathbb{E}[\#\{i, f(i, X_i) \leq \epsilon\}] \leq \epsilon. \tag{9.2}$$

A common way to build an NFA is to take, for each feature channel image, $f(i, \mathbf{x}) = N\mathbb{P}(X_i \geq \mathbf{x}_i)$, or

$$f(i, \mathbf{x}) = N\mathbb{P}(|X_i| \geq |\mathbf{x}_i|) \tag{9.3}$$

where X_i denotes a residual image. N is the number of tests, generally the overall number of pixels of all tested images.

Detection of anomalies of different sizes at a given scale. When working with colored noise, Grosjean and Moisan [GM09] recommend to convolve the noise with a measure kernel to detect spots of a certain size. This corresponds to the generation of new channels $\bar{r}(u) = r(u) * K$ where K is the normalized support of a disk of a given radius. But since we apply the detection at all dyadic scales, the tested radii are limited to a small set of N_{conv} values (1 to 3) at each scale. The disks are normalized to keep unit variance. Because the residual is assumed to be a stationary Gaussian field, the results after the filtering are also Gaussian. In that case, the test function is then

$$f(\bar{r}(u)) = N\frac{1}{2}\text{erfc}\left(\frac{|\bar{r}(u)|}{\sigma}\right) \tag{9.4}$$

with sigma the variance of $\bar{r}(u)$. The combination of the detections on the different versions of the residual is explicitly handled by the statistical test. Thus, the inputs to the detection phase are multichannel images of different scales, where each one is assumed to be a centered Gaussian field having unit variance at each pixel.

Multiscaling. The problem of anomaly detection is fundamentally multiscale. One would like to find both anomalies at a fine scale, for example small anomalies inside a texture, or at a coarser scale, for example a hole inside a textile. If one wants to be thorough, all these anomalies need to be detected whatever the scale they appear in. Even though the detection algorithm presented previously is not inherently multiscale, it can become so by applying it to all the different scales of the image. The idea to generate the different scales, process them and combine the result is

inspired by multiscale denoising such as [PMF17]. The scaled images can be computed by Gaussian subsampling. A Gaussian blur of parameter λ is applied to the image before a subsampling of 2 in each direction (so 4 in total). The standard deviation of the Gaussian λ is chosen so that the image has the same blur as the original one after subsampling. This blur is assumed to be of the order of 0.8 for natural well sampled images [Low04]. This means that the target blur before subsampling is $\lambda' = 2 \times 0.8$. We are searching for the blur λ such that the the Gaussian convolutions G verify

$$G_{\lambda'} = G_{\lambda}\left(G_{0.8}\right) \tag{9.5}$$

Since the variances of convolved Gaussians add up, this amounts to imposing a blur such that

$$\lambda'^2 = 0.8^2 + \lambda^2 \tag{9.6}$$

i.e.

$$0.8^2 + \lambda^2 = (2 \times 0.8)^2 \tag{9.7}$$

and therefore

$$\lambda = \sqrt{3} \times 0.8 \approx 1.39. \tag{9.8}$$

The process can then be iterated to compute all the dyadic scales. The corresponding pseudocode is in Algorithm 26.

Algorithm 26: $multiscaleDecomposition$: decompose an image in its different scales

input : An image u
output : A set (u_i) of images representing the different scales

$u_0 \leftarrow u$ // The first scale corresponds to the original image
for $i = 1$ *to* 3 **do**
    $u_i \leftarrow G_{1.39} * u_{i-1}$
        $G_{1.39}$ is a Gaussian kernel of standard
        deviation 1.39, the convolution is computed
        using a discrete filter of size $2\lfloor 4\sigma + 0.5 \rfloor + 1$
    $u_i \leftarrow subsample(u_i)$ // Subsample by a factor 4 (2 in each
        direction)

The statistical test. To detect anomalies for both sides of the tails, we use the NFA given in (9.3). It remains to compute the number of tests N which is significantly larger than the number of image pixels. Indeed, the detection occurs on N_{scales} different scales of the residuals, computed by Gaussian subsampling. The first scale is of the size of the original image, while each other scale is reduced by a factor of two. The produced residuals are of the size of their input images (in the case of the pixel method), or smaller (in the case of CNN features. See Section 9.3.3). Furthermore, the images have several channels and each test is replicated on all channels.

Denoting by Ω_s the set of pixels for the residual at scale s, N_{conv} the number of convolution filters and N_{chan} the number of channels, the number of tests is

$$N = N_{conv} N_{chan} \sum_{i=0}^{N_{scales}-1} |\Omega_s|. \tag{9.9}$$

In our case, N_chan corresponds to the number of channels of the image, N_{conv} to the number of radii tested and $\sum_{i=0}^{N_{scales}-1} |\Omega_s| = \frac{4u_h u_w}{3}$ with u_w and uh the width and the height the input

image. This leads to

$$N = \frac{4u_h u_w}{3} N_{chan} N_{conv} \tag{9.10}$$

The detection process is summarized in Algorithm 27.

Algorithm 27: *detect*: Detecting anomalies in a residual image

input : A residual image r, (u_w, u_h, u_c) the size of original image u (before any downsampling), the list R of disk radii to test, the NFA threshold ε, sp the patch size used for Alg. 24

output : A list of detections l

for R_i *in* R **do**
  Compute the disk kernel function f of radius R_i
  $\hat{r} \leftarrow r * f_i$ `// Convolve the image with the kernel (exact convolution done in Fourier)`
  Estimate σ the standard deviation of $\hat{r}$
  for *each pixel* $\hat{r}_k$ *of* $\hat{r}$ *outside of the border of size* $sp/2$ `// Detections are unreliable on the border`
  do
    if $\log\left(\frac{4u_h u_w u_c}{6} \text{length}(R) \text{erfc}\left(\frac{|\hat{r}_k|}{\sigma}\right)\right) < \varepsilon$ `// See Eqs (9.4) and (9.10)`
    then
      Add the position of the pixel and R_i to l

9.3.3 Choice of the image features

We now describe the image features, or channels, on which anomalies are detected. This is often considered an important step in anomaly detection. We found that our detection in the residual, which is unstructured noise, is fairly independent of the channel choice. We used with equal success the raw RGB colors as channels, or the intermediate feature channels of a pre-trained neural network. To that last purpose we used the VGG network [SZ15], a CNN trained on the ImageNet database [DDS+09].

We removed the network padding to guarantee spatial invariance. Indeed if the network padding is kept, the feature distributions differ at the borders of the feature maps [RDDM18]. We took the normalized version of the network and found slightly better results by working on the square root of the raw network features. Before the residual computation on NN features, we reduce their dimensionality with a PCA filter trained independently for each input image. This reduction is a compromise between the expressive power of the complete set of NN features and a compact image representation where visually almost-similar objects have similar representations. This transformation is represented on Figure 9.2.

9.3.4 Algorithm parameters

The main parameter of the statistical test is the number of allowed false alarms. In all of our experiments it was set to $1/100$. Hence an anomaly is detected at pixel $\mathbf{x}$ in channel i if and only if the NFA function $f(i, \mathbf{x})$ is below $\epsilon = 10^{-2}$. This means a theoretical expectation of less than 10^{-2} "casual" detection per image under the null hypothesis. We shall anyway observe much smaller NFAs for the real anomalies.

Another setting is the size of the disks for the convolutions. We got better results with disks of radius one and two for the basic method working on pixels, and radius one, two and three with

Neural Network features (See section 9.3.3). Neural Networks need bigger disks to remedy to the fact that medium-sized spots for the residual at a given scale may disappear at the following scale because Neural Networks features tend to ignore very small elements.

The other parameters are fixed as follows. We set the number of scales N_{scale} to 4 in all of our tests. The patch size for Algorithm 24 is set to $8 \times 8 \times 3$ for the basic version, while when using Neural Network features, we take the first five components with PCA and use a patch size of $5 \times 5 \times 5$. In both cases, the number n of patches for the search is set to 16. The similarity parameter h is set to 10.

9.4 Experiments

In this section we shall compare six methods methods:

- The [GM09] stochastic parametric background model. We denote this method by `Grosjean`.
- The [AT10] Fourier homogeneous model. We denote this method by `Aiger`.
- The [ZC10] non-local self-similar model. We denote this method by `Zontak`.
- The sparsity-based background model of [BCW14]. We denote this method by `Boracchi`.
- The non-local self-similar model of [MC14]. We denote this method by `Mishne`.
- Our non-local self-similar model [DEMD18]. We denote this method by `Davy`.

We use the NFA modified versions of these methods, presented in [EDMD19], whenever it is possible. The advantage is that the detection process is then only controlled by a single parameter making them easier to compare, especially for AUC estimation.

We propose two types of experimental comparison.

- The first comparison is a **qualitative** sanity check. For this qualitative analysis we tested on synthetic examples having obvious anomalies of different types (color, shape, cluster), or inexistent (white noise). These toy examples provide a sanity check since one would expect all algorithms to perform perfectly on them. We will also examine the results of the competitors on challenging examples taken from anomaly detection articles.
- The second protocol is a **quantitative** evaluation. We generated anomaly-free images as samples of colored random Gaussian noise. Being a spatially homogeneous random process, such images should remain neutral for an anomaly detector. We then introduced small anomalies to these images and evaluated whether these synthetic anomalies were detected by the competitors. This leads to evaluate a true positive detection rate (TP) for each method on these images. We also evaluated how much of the anomaly free background was wrongly detected, namely the false positive detection rate (FP). Disposing of TP-FP pairs yields ROC curves that will be opportunely discussed. Undoubtedly, the colored Gaussian noise used in this experiment could be replaced by any other spatially homogeneous random process. We varied the background texture by varying strongly the process's power spectrum.

9.4.1 Qualitative evaluation

The toy examples are probably the easiest to analyze. We show the results in Figure 9.9. We generated images in the classic form used in anomaly detection benchmarks like in [RMD+13], where the anomaly is the shape or the color that is unique in the figure. In the third toy example most rectangles are well spaced except in a small region. The anomaly therefore is a change in spatial density. Even though these examples are extremely simple to analyze, they appear to challenge several methods, as can be seen in Figure 9.9. Only [DEMD18] is able to detect accurately the anomaly in all three examples. This is explained in the second row where the residual after background subtraction is shown. In the residual details of the anomalies stand out on a noise-like background. While [AT10] works well with the color and the shape, it fails to detect the spatial density anomaly. [ZC10] detects well but also lots of false detection. The other methods [ZC10, MC14, GM09] and [BCW14] over-detect the contours of the non anomalous shapes, thus leading to many false positives. We also tried a sanity check with a pure white Gaussian noise image. This is done in the last two examples of Figure 9.9. [DEMD18], and [GM09] soundly detect no anomaly in white noise, as expected. However a few detections are made by [BCW14] and almost everything is detected by [MC14]. It can be noted that the background model of the first three papers is directly respected in the case of white Gaussian noise, which explains the perfect result. (In the case of the model of [DEMD18], it has to be noted that non-local means asymptotically transforms white Gaussian noise into white Gaussian noise [BCM08]). The over-detection in [MC14] can be explained by the lack of an automatic statistical threshold. The few spurious detections in [BCW14] show that the feature used for the detection doesn't follow a Gaussian distribution, contrarily to the method's testing assumption. It is also clear that one cannot build a sound sparse dictionary for white noise.

The same test was done after adding a small anomalous spot to the noise, and the conclusion is similar: [DEMD18, GM09] perform well, [BCW14] has a couple of false detections and doesn't detect the anomaly. One method, [ZC10], doesn't detect anything. Finally [MC14] over-detects. Both noise images were taken from [GM09].

We then analyze three examples coming from previous papers. The first one (first column in Figure 9.10) is a radar image of an undersea mine borrowed from [MC14]. The mine is detected by [DEMD18, GM09] without any false detections. Both [MC14, BCW14] have false detections; [ZC10] over-detects and [AT10] misses the mine. The second example (second column in Figure 9.9) shows an example of near-periodic texture. This is one of the examples where Fourier based methods are ideally well suited. It was therefore important to check if more generic methods were still able to detect the anomaly. Two methods [AT10] and [GM09] fail to detect the anomaly, the other three methods performing really well. This makes the case for self-similarity and sparsity based methods, that generalize nicely the background's periodicity assumption. The final example (third column from Figure 9.10) is a real example of medical imaging borrowed from [GM09] where the goal is to detect the tumor (the large white region). [AT10, BCW14] fail to detect the tumor. A strong detection is given by [ZC10, MC14] but the false alarms are also strong and numerous. Finally [DEMD18] has stronger tumor detections than [GM09]) (a NFA of $10^{-6.6}$ against $10^{-2.8}$) but it has several false alarms as well.

Finally we tested the methods on real photographs taken from the Toronto dataset [BT06]. This clearly takes several of the methods out of their specific context and type of images (tumors in X-ray images, mine detection in sonar scans, clot detection in microfibers, wafer defects,...) On the other hand, the principles of the algorithms are general. So by testing on these examples, our goal is to explore the limits of several detection principles, not to compare these specific algorithms. Clearly some of the methods are more adapted for spatially homogeneous background than to an outdoor cluttered scene.

Another issue when using real photographs is that anomalies detected by humans may be

semantic. None of the methods we consider was made to detect semantic anomalies, that can only be learned on human annotated images. Nevertheless, the tests' results are still enlightening. Detections are very different from one method to the other. The fourth example in Figure 9.10 shows a man walking in front of some trees. [AT10, GM09] and [MC14] don't detect anything. Both [ZC10,BCW14] detect mostly the trees and the transition between the road and the sidewalk. Surprisingly [DEMD18] only detects the man. Indeed in the noise like residual one can check that the man stands out. The second example shows a garage door as well as a brick wall. This time the algorithms tend to agree more. The conspicuous sign on the door is well detected by all methods as well the lens flare. A gap at the bottom between the brick wall and the door is detected by [DEMD18, MC14, GM09, BCW14]. The methods [MC14] and [BCW14] also detect the transition between the wall and and the brick wall. Finally some detections on the brick wall are made by [DEMD18] and [BCW14]. The residuals of [DEMD18] on the second row are much closer to noise than the background, which amply justify the interest of detecting on the residual rather than on the background. Nevertheless, the residual has no reason to be uniform, as is apparent in the garage's residual. Even if the detections look any way acceptable, this non-uniformity of the residual noise suggests that center-surround detectors based on a local variance (as done in [GM09]) might eventually be preferable.

Fixing a target number of 10^{-2} for the NFA means that under the $(\mathcal{H}_0)$ model, only 10^{-2} false positives should occur per image. Yet, many of them shown examples show several false positives. Given the mathematical justification of these thresholds, false positives come from discrepancies between the hypothetical $(\mathcal{H}_0)$ model and the image. In the case of [ZC10], the over-detection in the trees of the picture with a man can be explained by the limited self-similarity of the trees: for this region, the nearest patches won't be close enough to the patch to reconstruct to fit the model, which requires at least one would-be-identical-except-for-the-noise patch in the neighborhood. The over-detection in the case of the undersea mine is likely a mismatch of the noise model with the picture noise. The many false alarms of this method for the other examples, makes us wonder if the model hypothesis is not too strong. The [BCW14] method triggers many false detections in almost all examples tested. As we mentioned, this suggests that the Gaussian model for the detection pairs is inaccurate. This is not necessarily a problem for specific fault detection applications where the false alarm curves can be learned.

9.4.2 Quantitative evaluation

Estimating how well an anomaly detector works "in general" is a challenging evaluation task. Qualitative experiments such as the ones presented in section 9.4.1 give no final decision. Our goal now is to address the performance evaluation in terms of true positive rate (TP) and false positive rate (FP). To that aim, we generated a set of ten RPN textures [GGM11a] which are deprived of any statistical anomalies. We then introduced one artificial anomaly per rpn by merging a small piece of another image inside each of them. This was made by simple blending or by Poisson editing [PGB03] using the implementation of [DMFML16]. This method provides a set of images where a ground truth is known. Hence the detection quality measure can be clearly defined. Figure 9.5 shows one of the generated RPN images with an anomaly added and the anomaly's ground truth locus. Table 9.1 shows the result for our six methods on this dataset.

Table 9.1 demonstrates that for all methods, the predicted number of false positives (namely the theoretical NFA) is not always achieved. Indeed, the threshold for Table 9.1 was chosen so that the theoretical number of false detections per image should be 10^{-2}. When taking into account the total number of pixels, this means that only around $4 \times 10^{-6}\%$ false detections should be made by any method in this table. Only two methods are close to this number: [AT10] and [DEMD18], while the other compared methods make too many false detections. Such a false

Figure 9.5: A ground truth (on the right) for anomaly detection has been generated by introducing an anomaly in a RPN [GGM11b] texture (on the left), which is anomaly free. The detection is then done on the result (in the middle).

	TP pixels (in %)	FP pixels (in %)	TP anomalies (in %)	FP anomalies (in %)
[AT10]	56.2	7.60×10^{-4}	90	40
[ZC10]	0	0	0	0
[MC14]	23.4	8.52	90	90
[BCW14]	78.2	0.87	100	100
[GM09]	11.6	0.16	30	20
[DEMD18]	33.1	1.79×10^{-5}	80	10

Table 9.1: Quantitative comparative results for anomaly detection. The number of true positive (TP) and false positive (FP) for different metrics is shown. TP pixels and FP pixels correspond to detections at a pixel level. A true positive is when an anomalous pixel is detected, and a false positive when a normal pixel is detected as anomalous. TP anomalies and FP anomalies evaluate if anomalies have been detected at all. A true positive is counted when there is at least one detected pixel in an anomalous region, and a false positive when there is at least one detection completely outside an anomalous region (with a maximum of 1 FP per image). These results were computed on a dataset of random uniform textures with a single anomaly added to each image. The thresholds were set for a targer number of false alarms (NFA) of 10^{-2} per image (theoretical FP pixels of $4 \times 10^{-6}\%$). An example of an image from the dataset is shown in Figure 9.5. A method works correctly if it detects a high percentage of anomalies (third column) while having a good pixel coverage (first column), and a minimal false positive rate (second and fourth columns). Having a very low false positive rate is crucial for massive fault detection. In that sense, the best methods are [AT10] and [DEMD18].

positive target might seem too strict. Yet, it is an important requirement of anomaly detectors in fault detection to minimize the false alarm rate. Indeed excessive false alarms may put a production chain in jeopardy. Images are generally of the order of 10^7 pixels. Therefore if one wants to limit the false detection rate in a series of tested images, the false positive rate needs to be really small. The methods compared - except [MC14] - used the NFA framework. Therefore the discrepancy between the theoretical target and the obtained number of false alarms is explained by an inadequate $(\mathcal{H}_0)$ for the images. In fact, only the background model of [AT10] matches completely these really specific textures that are RPNs.

To better compare the methods, we also computed ROC curves for all methods, Figures 9.7 and 9.8, as well as the table of true positive areas and false positive areas for a fixed positive rate of 1% (Table 9.2). The ROC curve aren't impacted by the choice of thresholds. Figure 9.7 is shown with a log scale for the number of false positives because its low or very low false positive section is much more relevant for anomaly detection than the rest. From these ROC curves and tables we can conclude, for this specific example, that [AT10] (Area under the curve (AUC) 7.52) (which theoretically should be optimal for this problem) performs the best followed closely by [DEMD18] (AUC 7.03). It's worth noting that [DEMD18] is performing better than [AT10] for very low false positive region. We then have [BCW14] (AUC 5.79). The trailing methods

	TP pixels (in %)	FP pixels (in %)	TP anomalies (in %)	FP anomalies (in %)
[AT10]	79.1	1.0	100	100
[ZC10]	27.2	1.0	60	60
[MC14]	12.5	1.0	50	90
[BCW14]	80.1	1.0	100	100
[GM09]	24.2	1.0	70	100
[DEMD18]	65.0	1.0	100	100

Table 9.2: This table is similar to Table 9.1, but in this case each method detection threshold is set so as there are 1% false positives. Hence, the criterion is to detect as many anomalies as possible (third column) while having a high true positive rate. The winners are clearly [BCW14] and [AT10].

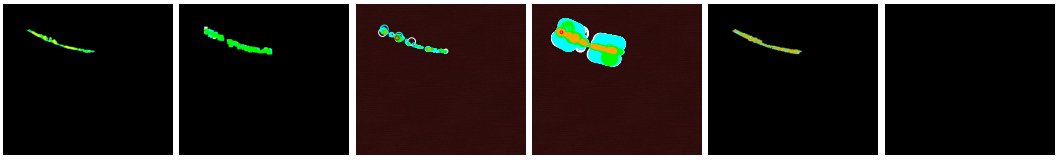


Figure 9.6: Example of detections for all the different methods on 9.5. It corresponds to the one showed in Table 9.1. From left to right: [AT10], [BCW14], [DEMD18], [GM09], [MC13] and [ZC10].

are [GM09] (AUC 3.30), [ZC10] (AUC 2.92) and finally [MC14] (AUC 1.98) . Nevertheless, if a moderate number of false positives can be tolerated, then [BCW14] becomes really attractive because of its high detection precision. Figure 9.8 illustrates the problem of false detections. Most methods requires many false detections to achieve a reasonable detection rate. Only [AT10] (AUC 0.82) and [DEMD18] (AUC 0.87) detect well while still keeping a zero false detection rate. This confirms the results from Table 9.1. Table 9.2 also shows that having a 1% detection is useful to obtain a good precision but leads to almost all images getting false positives. In practice 1% is too large a tolerance for images. In Figure 9.6 we show the result of the detections on 9.5 corresponding to Table 9.1 for the different methods.

9.4.3 Computation time analysis

In this section we do a brief computation time analysis. All algorithms have wildly different computation times. For example [AT10] method is really fast as no really complex computations are needed. On the contrary the [MC14] method is really slow. Table 9.3 summarizes the computation time for the different algorithms for the parameter used for the experiment. It's worth noting that for the larger parameters the [MC14] method requires many hours to compute a single result. It's also worth noting that even though the [BCW14] and [DEMD18] algorithms are not the fastest ones, the dictionaries of patches and indexes for the searches can be precomputed and therefore accelerated for fast industrial applications. For example the processing of [BCW14] only takes 12s when the dictionary is prelearned. The computation time estimation was done on a core i7-7820HQ 2.90GHz using authors' code whenever it was available ([BCW14], [DEMD18] and [MC14] are multithreaded so actual computation times are reported. We report $1/8$ of the actual computation time for [AT10], [GM09] and [ZC10] for a fair comparison).

[AT10]	[BCW14]	[DEMD18]	[GM09]	[MC14]	[ZC10]
0.09	1375	57	1.4	749	394

Table 9.3: Computation time (in seconds) for the different methods reviewed in details with the parameter chosen for the experiments for the door image (size: 600×450).

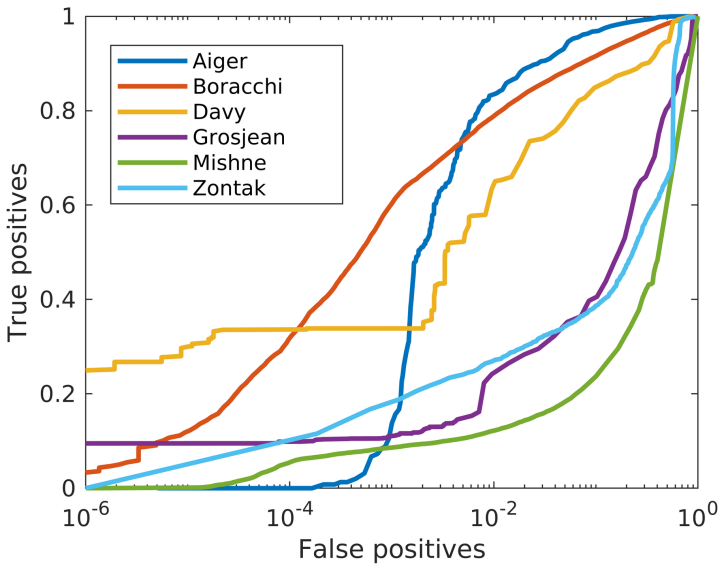


Figure 9.7: ROC curve computed on the dataset of synthetic images. A true positive corresponds to an anomalous pixel detected. A false positive corresponds to a normal pixel that has been detected as anomalous. In deep blue [AT10] (Area Under the Curve (AUC) 7.52), in red [BCW14] (AUC 5.79), in yellow [DEMD18] (AUC 7.03), in purple [GM09] (AUC 3.30), in green [MC14] (AUC 1.98) and in light blue [ZC10] (AUC 2.92).

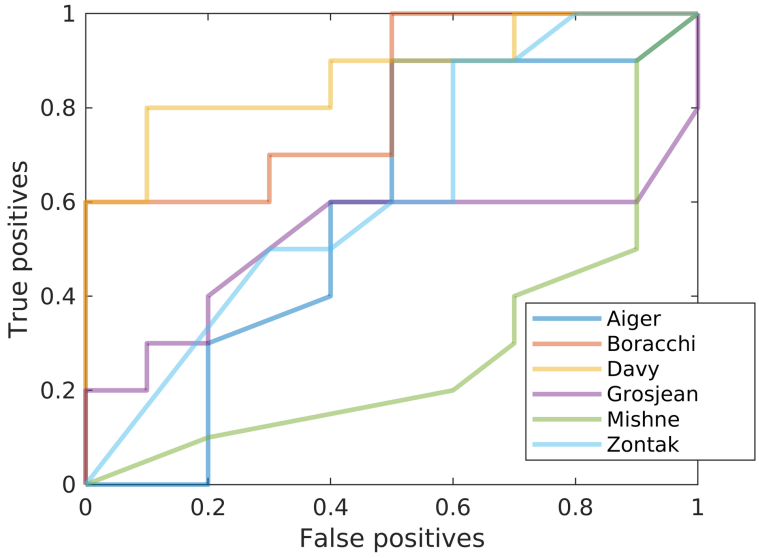


Figure 9.8: ROC curve computed on the dataset of synthetic images. A true positive is when an anomaly is detected (in the sense that at least one detection has been made inside the anomalous region). A false positive is when there is a detection outside the anomalous region. In deep blue [AT10] (Area Under the Curve (AUC) 0.82), in red [BCW14] (AUC 0.585), in yellow [DEMD18] (AUC 0.87), in purple [GM09] (AUC 0.52), in green [MC14] (AUC 0.28) and in light blue [ZC10] (AUC 0.625).

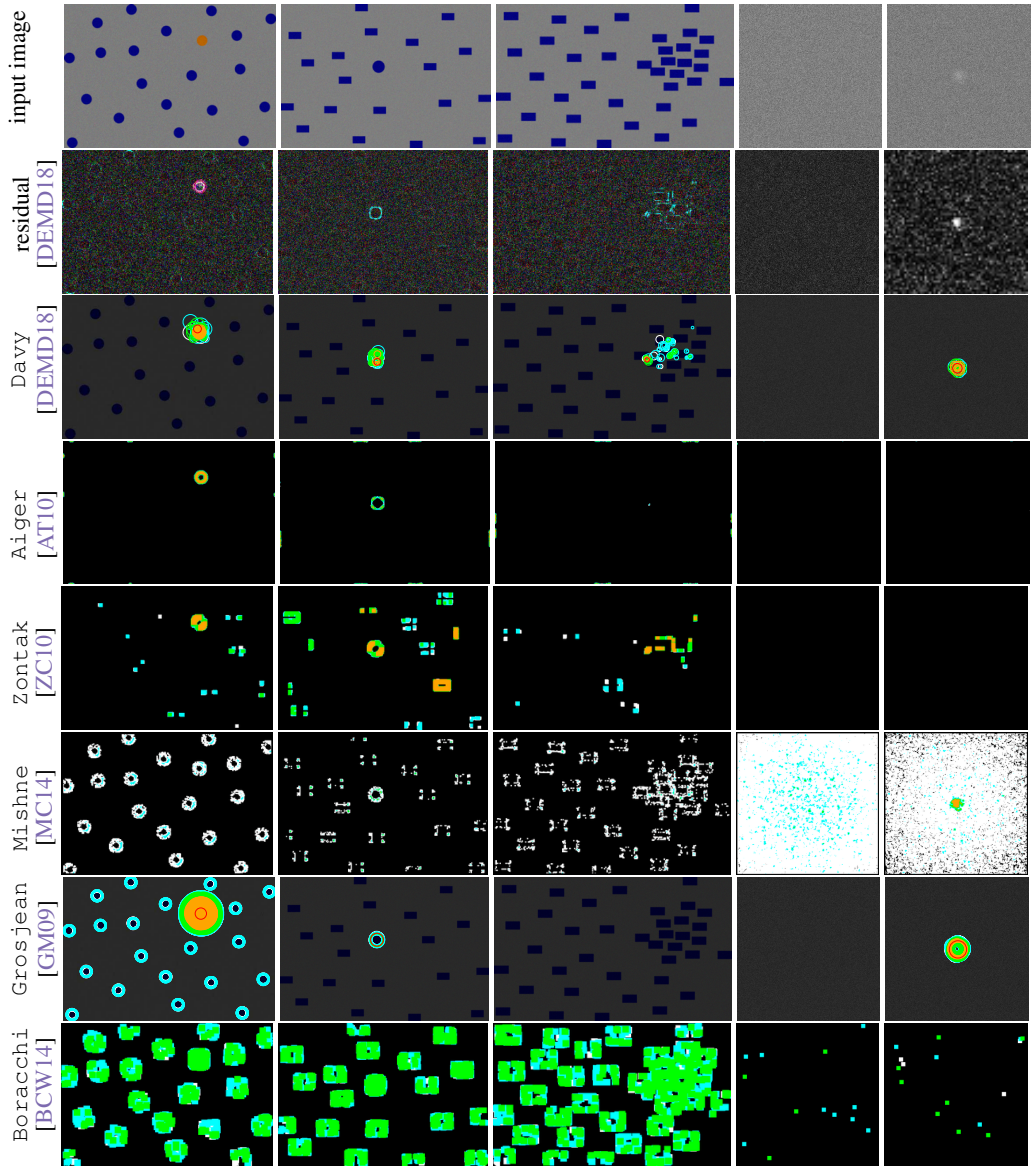


Figure 9.9: From left to right: Image presenting an anomaly in colors, in shape and in density, image of pure noise, and image of noise with an anomaly in the middle (from [GM09]). From top to bottom: The original image, the image residual of one of the scales computed in [DEMD18] (the scale shown is the one where the anomaly is the most salient, the contrast has been adjusted for visualization purpose), algorithm detections for: [DEMD18], [AT10], [ZC10], [MC14], [GM09] and [BCW14]. Detections are shown using the following color coding: white is a weak detection - threshold with NFA $\in [10^{-3}, 10^{-2}]$, cyan is a mild detection - threshold with NFA $\in [10^{-8}, 10^{-3}]$, green is a strong detection - threshold with NFA $\in [10^{-21}, 10^{-8}]$, and orange is very strong - threshold with NFA $\leq 10^{-21}$. When available, red is the detection with the threshold corresponding to the lowest NFA. For [MC14] we adopted a similar color coding: white between 0 and 0.5, cyan between 0.5 and 0.7, green between 0.7 and 0.9 and orange above 0.9.

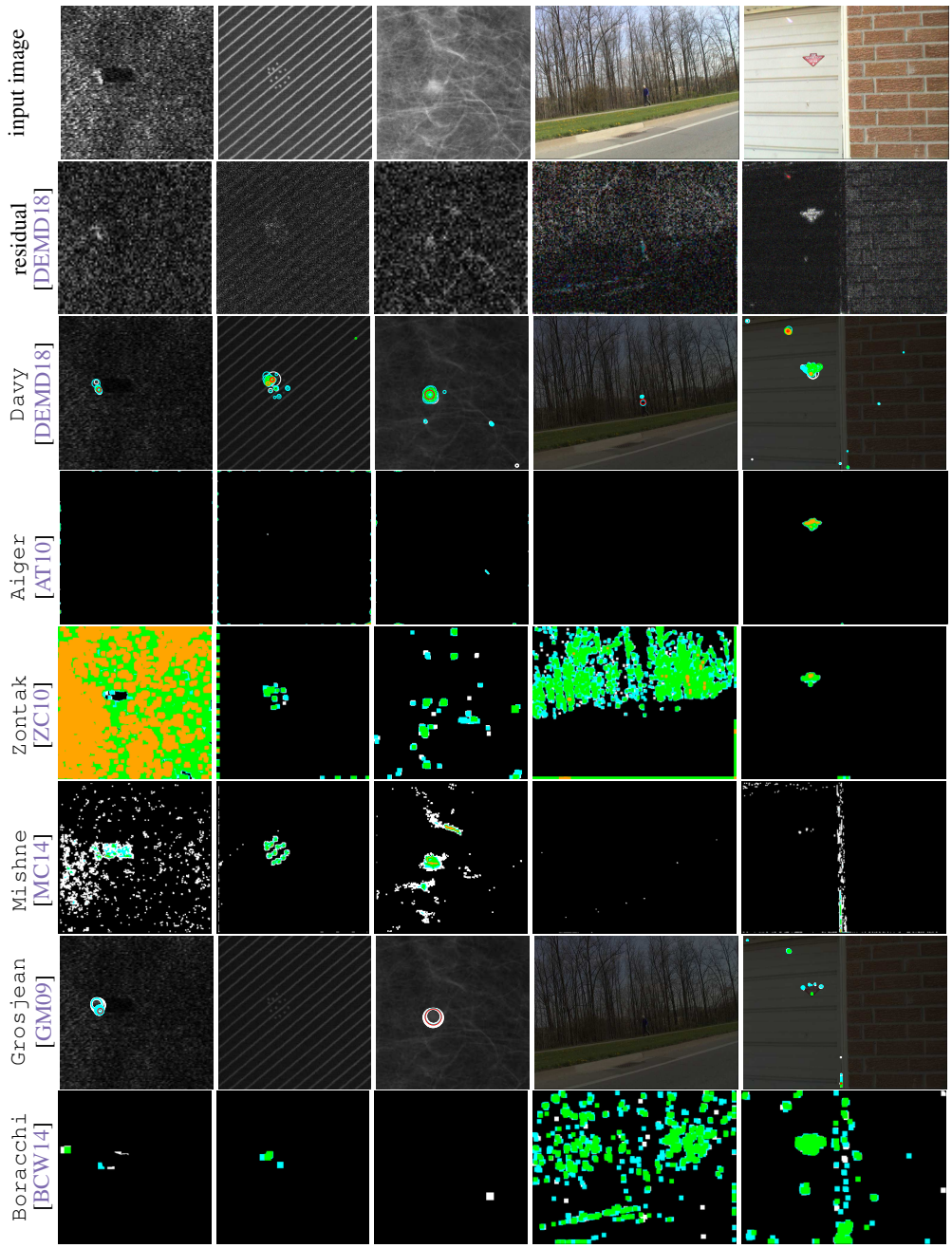


Figure 9.10: From left to right: image of an undersea mine from [MC14], image of a periodic textile from [TH99], image of a tumor from [GM09], image of a man from the Toronto dataset [BT06], image of a garage door from [BT06]. From top to bottom: The original image, the image residual of one of the scales computed in [DEMD18] (the scale shown is the one where the anomaly is the most salient, the contrast has been adjusted for visualization purpose), algorithm detections for: [DEMD18], [AT10], [ZC10], [MC14], [GM09] and [BCW14]. Detections are shown using the following color coding: white is a weak detection - threshold with NFA $\in [10^{-3}, 10^{-2}]$, cyan is a mild detection - threshold with NFA $\in [10^{-8}, 10^{-3}]$, green is a strong detection - threshold with NFA $\in [10^{-21}, 10^{-8}]$, and orange is very strong - threshold with NFA $\leq 10^{-21}$. When available red is the detection with the threshold corresponding to the lowest NFA. For [MC14] we adopted a similar color coding: white between 0 and 0.5, cyan between 0.5 and 0.7, green between 0.7 and 0.9 and orange above 0.9.

9.5 Conclusions

We shall now address the objections that come to mind.

Is the method new? We have listed several saliency or anomaly detection methods based on rarity (sparsity, lack of similar patches, etc.) Thus involving sparsity or self-similarity is not new. The novelty of the method seems to be that it builds a new image, the residual, where the self-similar structure has been eliminated. As we have seen, using the *a contrario* framework is not new, but its use was restricted to detection in Gaussian colored noise.

Is the residual really stationary noise? The hypothesis is that the residual is noise, in the sense that it has lost all self-similarity. But this noise might not be stationary, which would lead to detection misses. Indeed, as a toy example assume that an image is composed of two textures, one very contrasted, and the other not. Then, the residual will have higher variance in the contrasted part. If the anomaly lies in the non-contrasted region, it might be missed because the (global) noise variance is overestimated (for this region). This can only be solved by localizing the detection, namely estimating the noise variance more locally, or equalizing its variance to make it stationary. This remains to be investigated.

Redundant detections? We have shown the detector performance either on the color channels or on VGG feature channels. Observe that these detections can be fused by a mere union, as they are all meaningful. They are, as we saw, redundant and were shown to illustrate the independence of the method from the chosen channel.

Extensions. An extension to video is highly desirable but requires a computationally intensive implementation to perform nonlocal space-time patch comparison.

10 Analysis of PatchMatch

Many problems in image/video processing and computer vision require the computation of a dense k-nearest neighbor field (k-NNF) between two images. For each patch in a query image, the k-NNF determines the positions of the k most similar patches in a database image. With the introduction of the *PatchMatch* algorithm, Barnes et al. demonstrated that this large search problem can be approximated efficiently by collaborative search methods that exploit the local coherency of image patches. After its introduction, several variants of the original *PatchMatch* algorithm have been proposed, some of them reducing the computational time by two orders of magnitude. In this chapter we study the convergence of *PatchMatch* and its variants, and derive bounds on their convergence rate. We consider a generic *PatchMatch* algorithm from which most specific instances found in the literature can be derived as particular cases. We also derive more specific bounds for two of these particular cases: the original *PatchMatch* and Coherency Sensitive Hashing. The proposed bounds are validated by contrasting them to the convergence observed in practice. This work was published in [EA18].

10.1 Introduction

Patch-based methods are among the state-of-the-art in several image/video processing and computer vision applications. Often these methods require finding for all patches of a query image, the (approximate) k nearest neighbors among the set of patches of a database image. This is referred to as an *approximate k nearest neighbors field* (k-ANNF) from the query image to the database image.

Examples of the application of k-ANNFs can be found for image completion (and editing) [AFCS11,BSFG09a], denoising of images [BSGF10] and video [BZZM15,LF10], video stylization [BZL+15, BZZM15], alpha matting [HRR+11], optical flow [BYJ14, FBK15, HBK+14, LYMD13] and stereo-vision [LYMD13, BRR11]. Also in the close field of computer graphics ANNFs of 3D surface patches have been applied to mesh tracking [KH11] and texture transfer [CFGS12].

The brute-force computation of the k-NNF scales linearly with the product of the number of pixels in the query and the database images, and is therefore prohibitively slow. The first practical approaches rely on data structures such as hash tables or partition trees (see [KZN08] and references therein). The approximate k nearest neighbors of each query patch are computed independently. Even if these approaches greatly improve with respect to the brute-force search, they are still too slow (and moreover scale badly with the patch size) for many applications such as those requiring user interaction.

The introduction of the *PatchMatch* algorithm [BSFG09a] and its generalized version

[BSGF10] has represented a breakthrough in the field. It brings a speed-up of almost two orders of magnitude over previous search techniques. The main reason for this is that *PatchMatch* performs all queries simultaneously and in collaboration, exploiting the fact that overlapping query patches are likely to have overlapping matches in the database image.

PatchMatch is an iterative algorithm which starts from a random initial guess for the k-ANNF and gradually refines it. In each iteration, each patch *propagates* its current candidates to its neighboring patches on the query image grid. Additionally, new candidates are searched randomly in the database image. This is performed by uniformly sampling in a series of concentric squares with decreasing radius centered at the position of the current best candidate.

Several works have improved the original *PatchMatch* algorithm reporting gains of one order of magnitude or even more. The main theme of these works is to combine *PatchMatch* with classical search structures to improve the random initialization or the random search. *Coherency Sensitive Hashing* (CSH) [KA11] uses locality sensitive hashing [IM98] to improve both the propagation and the random search steps. KD-trees [Ben75] are used in [OA12] instead of the random initialization (after reducing the dimensionality of the patches). In [HS12] a KD-tree is used for the random search.

In this chapter we extend and generalize the theoretical framework of [ACF12] formalizing *PatchMatch*-like techniques. We apply it to study their convergence and give upper bounds on their convergence speed which are tighter than the ones of [ACF12]. In particular we study the original *PatchMatch* [BSFG09a], its generalization to k nearest neighbors [BSGF10] and CSH [KA11]. Our estimates of a geometric convergence rate confirm the intuitions that led to the design of these algorithms. They also provide insight that might help improving current techniques.

We interpret *PatchMatch* in a general setting, as a collaborative optimization tool. While the results in [ACF12] consider only the original *PatchMatch* algorithm with one nearest neighbor, our results apply to k-nearest neighbors and to more complex *PatchMatch* algorithms, such as CSH [KA11], propagation-assisted KD-trees [HS12] and RIANN [BZZM15].

We start by giving a precise definition of a generic *PatchMatch* algorithm in §10.2. In §10.3 we analyze the convergence of the generic *PatchMatch*, by bounding the probability of having energies higher than a given threshold after an iteration of the algorithm. In §10.4 we consider two specific algorithms, the original *PatchMatch* [BSFG09a] and CSH [KA11] and derive specialized bounds for them. Finally, in §10.5 we compare the theoretical bound to empirical cases. The proofs of our results can be found in the supplementary material.

10.2 Generic fast patch matching algorithm

10.2.1 Notations

We denote the query image by $\mathbf{A}$ and the database image by $\mathbf{B}$. We write $x \in \mathbf{A}$ if $x \in \mathbb{R}^d$ is a patch of the image $\mathbf{A}$ (with this abuse of notation, $\mathbf{A}$ is both an image and the set of its patches). The k-NNF from image $\mathbf{A}$ to $\mathbf{B}$ is denoted by φ. For a patch $x \in \mathbf{A}$ the matches associated to x in $\mathbf{B}$ are written as φ_x, which is a set of k distinct patches of $\mathbf{B}$.

Definition 10.1 (Matching energy)**.** *The quality of the matching of a patch $x \in \mathbf{A}$ is measured by an the energy function $U_x(.)$ defined for any patch $z \in \mathbb{R}^d$ with values between 0 and $+\infty$. We generalize this definition to a set of patches u by*

$$U_x(u) = \max_{y \in u} U_x(y). \tag{10.1}$$

This means that the minimum value of U_x over a set of k elements without repetitions is 0.

In practice, the L_2 distance in $\mathbb{R}^d$, written as $\|.\|_2$, is often used to define as matching energy. To have a minimum value of 0 over k-sets, we write it as

$$U_x(z) = \|x - z\|_2 - K_x \tag{10.2}$$

where if $N_k(x) \in \mathbf{B}$ is the actual k^{th} nearest neighbor of x in $\mathbf{B}$, $K_x = \|x - N_k(x)\|_2$. It follows that if u is a set of k distinct patches of $\mathbf{B}$, then $U_x(u) \geqslant 0$. We write $\overline{\varphi}_x$ to denote the worst match in this set, defined by

$$\overline{\varphi}_x = \arg\max_{y \in \varphi_x} U_x(y), \tag{10.3}$$

where $U_x(.)$ is the energy function of Definition 10.1.

The k-NNF φ assigns to each $x \in \mathbf{A}$ a k-set φ_x that minimizes U_x, i.e. such that $U_x(\varphi_x) = 0$. The goal of *PatchMatch* algorithms is to find an approximate solution to this optimization problem. In the following we define a level-set, written $\{U_z \geqslant \alpha\}$, for η a k-set of elements $\eta \in \{U_z \geqslant \alpha\}$ if and only if $U_z(\eta) \geqslant \alpha$.

The following operator selects the k patches with smaller energy U_x from a larger set.

Definition 10.2 (Merge operator)**.** *We define* merge_x^k *as an operator transforming a set of more than* k *patches into a set of exactly* k *patches satisfying:*

$$|\text{merge}_x^k(u)| = k \qquad \textit{and} \qquad \forall p \in u \backslash \text{merge}_x^k(u), U_x(p) \geqslant U_x(\text{merge}_x^k(u)). \tag{10.4}$$

We use basic graph notation in the next sections. Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ the set of edges. By $y \sim x$ we denote that y is parent of x, *i.e.* $(y, x) \in \mathcal{E}$. We finally write $\mu(x)$ for the number of parents of a node x, i.e.

$$\mu(x) = |\{y \mid y \sim x\}|. \tag{10.5}$$

A path of size m from x to z corresponds to a set of vertices $(c_i)_{i=1,\dots,m} \in \mathcal{V}^m$ such that $c_1 = x$, $c_m = z$ and for all $i \in [\![1, m-1]\!]$, $(c_i, c_{i+1}) \in \mathcal{E}$. We write $\mathcal{P}(x, z)$ for the set of all paths (of any size) from x to z.

10.2.2 Propagation graph

In a *PatchMatch* algorithm all patches in $\mathbf{A}$ collaboratively search for their matches in $\mathbf{B}$. This collaboration is achieved through the propagation of candidate matches from each query patch to other patches in $\mathbf{A}$ following a specific order. This process can be described by defining a directed acyclic graph over the set of patches of $\mathbf{A}$, which we call the *propagation graph* following [ACF12]. The vertices $\mathcal{V}$ in the propagation graph $\mathcal{G}$ are the set of patches of $\mathbf{A}$, and the directed edges $\mathcal{E}$ describe the propagation relationships: $(z, x) \in \mathcal{E}$ means that z propagates matches to x. We associate an action function to each edge to allow for the possibility of applying a transformation to the propagated matches.

Definition 10.3 (Propagation action)**.** *An action associated to an edge* $e \in \mathcal{E}$ *in the propagation graph, written* A_e*, is a function which takes as argument a patch of an image and returns a patch. This patch can either be from the same image or be a new one generated by the action.*

The vertices in the propagation graph are indexed using a topological ordering (such an ordering exists because the graph is acyclic). The propagation follows this ordering. The complete definition of the propagation graph is the following, and it fully specifies the propagation in a *PatchMatch* algorithm.

Definition 10.4 (Propagation graph)**.** *A propagation graph, written $\mathcal{G}$, corresponds to a triplet $(\mathcal{V}, \mathcal{E}, A)$ where $\mathcal{V}$ is the set of patches of image $\mathbf{A}$, $\mathcal{E}$ a set of edges such that the graph is connected and acyclic and A are the action functions associated to each edge (Def. 10.3).*

For matching image patches, the forward propagation follows the raster order from the top-left to the bottom-right. Each patch propagates matches to its neighbors on the right and down:

$$\mathcal{E} = \{(x, y) \in \mathcal{V} \times \mathcal{V} \mid y = \text{right}(x) \text{ or } y = \text{down}(x)\}\,. \tag{10.6}$$

The action shifts the candidate patch following the direction of propagation: if $x \in \mathbf{A}$ has a candidate $z \in \mathbf{B}$, it propagates $\text{right}(z) \in \mathbf{B}$ to $\text{right}(x)$. Thus, $A(x, \text{right}(x)) = \text{right}(\cdot)$, and analogously $A(x, \text{down}(x)) = \text{down}(\cdot)$.

Some variants of *PatchMatch* add additional edges to this basic propagation graph seeking to enhance the impact of the propagation [BSGF10, KA11]. These additional transitions connect pairs of patches in $\mathbf{A}$ that are similar, and thus good matches for one of patches in the pair are natural candidates for the other. The action associated with these new edges is simply the identity function.

PatchMatch algorithms have also been applied on meshes [KH11, CFGS12]. In those cases, the propagation can be defined by considering a DAG on a subgraph of the mesh.

10.2.3 Random search

For the specification of a *PatchMatch* algorithm we need a mechanism for gathering random samples around the current candidates.

Definition 10.5 (Random sampling operator)**.** *Given a database image $\boldsymbol{B}$ we define the transition kernel Q such that for any k-set φ of patches from image $\boldsymbol{B}$, $Q(\varphi, \cdot)$ is a probability on the k-sets of patches from $\boldsymbol{B}$. A set of k patches drawn from $Q(\varphi, \cdot)$ will be denoted by $S\varphi$, i.e. $S\varphi \sim Q(\varphi, \cdot)$. We consider transition kernels with the property of having a non-zero probability of transitioning to a k-set of matches with arbitrary positive energy:*

$$Q(\varphi, \{U_x < \varepsilon\}) > 0, \qquad \forall \varphi, \forall x \in \boldsymbol{A}, \forall \varepsilon > 0. \tag{10.7}$$

Defined alongside the kernel Q is the *worst case transition probability* for an energy level ε at $z \in \mathbf{A}$, as the highest probability of transitioning between two sets of patches with energy level higher than ε:

$$C(z, \varepsilon) := \sup_{\eta \in \{U_z \geqslant \varepsilon\}} Q(\eta, \{U_z \geqslant \varepsilon\}). \tag{10.8}$$

Lemma 10.1. *For all $z \in \boldsymbol{A}$, $C(z, .)$ is a non-increasing function such that for $\varepsilon < 0$, $C(z, \varepsilon) \in [0, 1[$ and $C(z, \varepsilon) = 1$ for $\varepsilon \leqslant 0$.*

10.2.4 Algorithm

The generic *PatchMatch* is presented in Algorithm 28. It starts by initializing the propagation graph and the candidate matches. The propagation graph is initialized by defining a set of edges $\mathcal{E}$ and the associated propagation actions A.

The core of the algorithm is the iterative process that cycles through the nodes in the topological ordering updating the candidate matches according to the steps 5 and 8. The update in step 5 is the result of selecting the best k matches from a set of candidates given by: the current k matches φ_x^n; the matches $A_{y,x}\varphi_y^{n+1}$ propagated from the parent nodes $y \sim x$ in the propagation graph; and samples $S_2 A_{y,x}\varphi_y^{n+1}$ gathered around them. We call the latter set of samples the *randomized*

propagation. In step 8 k random samples $S_1\varphi_x^{n+1/2}$ around the new current matches are drawn to finish the update process.

The random samplings S_1 and S_2 are drawn from transition kernels Q_1 and Q_2 which might differ. The parent nodes $y \sim x$ precede x in the topological ordering, therefore they propagate to x the updated candidates φ_y^{n+1}.

After each iteration the propagation graph is reversed. The last step considers the possibility of modifying the propagation graph by adding/removing relevant/irrelevant edges. This is done for example in CSH [KA11].

Algorithm 28: Generic patch matching algorithm

1 Initialize propagation graph $\mathcal{G}$

2 Initialize matching φ^0
3 **for** $n \in \mathbb{N}$ **do**
4 **Update candidates**
5 **for** $x \in \mathcal{V}$ *following the topological ordering* **do**
6 $\varphi_x^{n+1/2} = \text{merge}_x^k \left(\varphi_x^n \cup \bigcup_{y \sim x} A_{y,x}\varphi_y^{n+1} \cup \bigcup_{y \sim x} S_2 A_{y,x}\varphi_y^{n+1} \right)$
7
8 $\varphi_x^{n+1} = \text{merge}_x^k \left(\varphi_x^{n+1/2} \cup S_1\varphi_x^{n+1/2} \right)$
9 **Reverse propagation graph**
10 **Update propagation graph**

10.3 Convergence of the patch matching algorithms

In this section we study the convergence of the generic *PatchMatch* algorithm described in the previous section. We do so by upper-bounding the probability that the energy U_x at a node x is larger than a threshold ε after an iteration of *PatchMatch*.

10.3.1 Point-wise energy decay

Our main result is Theorem 10.1 which bounds the decay of the probability of $U_x(\varphi_x^{n+1} > \varepsilon)$. The propagation makes this probability smaller, since any of the ancestors of x could find a candidate that when propagated to x would yields an energy below ε. Indeed, as a consequence of the propagation, if φ_x has energy higher than ε after a propagation pass, then the energies of the ancestors z of x need to be higher than a series of levels ε_{zx}. The violation of any of these restrictions would cause the propagation of a candidate match to x with energy U_x below ε. The higher these levels, the smaller the probability of not sampling random candidates with energies below them, and thus the smaller $P(U_x(\varphi_x^{n+1}) > \varepsilon)$.

In Lemma 10.2 we show that imposing a lower bound ε of the matching energy of a node x results in lower bounds $\varepsilon_{z,x}$ for all its ancestors z that can be calculated recursively starting from x and following the reversed propagation ordering. In Theorem 10.1 we bound the probability that none of x's ancestors draw a random candidate with energy lower than the corresponding $\varepsilon_{z,x}$.

Lemma 10.2 (Constraints propagation)**.** *Consider an assignment φ^{n+1} resulting from an iteration of Algorithm 28. Then for each pair of nodes $x, z \in \mathcal{V}$,*

$$U_x(\varphi_x^{n+1}) \geqslant \varepsilon \Rightarrow U_z(\varphi_z^{n+1}) \geqslant \varepsilon_{z,x}, \tag{10.9}$$

where the levels $\varepsilon_{z,x} \geqslant 0$ are as follows. For the ancestors of x (i.e. $\mathcal{P}(z,x) \neq \emptyset$) the levels $\ell^{\varepsilon}_{z,x}$ are defined via the following recursion starting from x and following the inverse propagation order:

$$\begin{cases} \varepsilon_{z,x} = \min \left\{ U_z(\theta) \;\middle|\; \theta \in \bigcap_{y \text{ s.t. } z \sim y} A_{z,y}^{-1}(\{U_y \geqslant \varepsilon_{y,x}\}) \right\} \\ \varepsilon_{x,x} = \varepsilon. \end{cases} \tag{10.10}$$

For the rest of the nodes $\varepsilon_{z,x} = -1$.

Once we have established this "allowed" sets for the ancestors of x, the idea of the proof is to determine the probability of not escaping the allowed sets in any of the random searches.

Suppose z is an ancestor of x with a candidate $\varphi_z^n \in \{U_z \geqslant \varepsilon_{z,x}\}$ (for simplicity assume $k = 1$). The probability of keeping the energy higher than ε after one iteration decreases because there is a non-zero probability of taking a random sample outside this level set. The probability of sampling a candidate in an upper-level set is

$$\mathbb{P}(S\varphi_z^n \in \{U_z \geqslant l\} \mid \varphi_z^n \in \{U_z \geqslant l\}) = \frac{1}{\int_{\{U_z \geqslant l\}} \mathbb{P}(d\varphi_z^n)} \int_{\{U_z \geqslant l\}} Q(\varphi_z^n, \{U_z \geqslant l\}) \mathbb{P}(d\varphi_z^n). \tag{10.11}$$

If we knew the probability distribution of the candidate φ_x^n at iteration n, we could compute the above probability exactly. Instead, we bound it by assuming that all the mass of the distribution of φ_z^n concentrates on a single point: the one from which it is more unlikely to draw a sample with energy lower than l. The resulting probability is given by the worst case transition probability C in (10.8). This is the main intuition in the proof of our main result.

Theorem 10.1 (Point-wise convergence)**.** *Consider the field of candidate matches at iteration n, φ^n. Define φ^{n+1} by applying an iteration of the Generic PatchMatch in Algorithm 28. Then, for all $\varepsilon > 0$, for all $x \in \mathbf{A}$, we have*

$$\mathbb{P}(U_x(\varphi_x^{n+1}) \geqslant \varepsilon) \leqslant \mathbb{P}(U_x(\varphi_x^n) \geqslant \varepsilon) \prod_{z \in \mathbf{A}} \left(C_2(z, \varepsilon_{z,x})^{\mu(z)} C_1(z, \varepsilon_{z,x}) \right), \tag{10.12}$$

where $\mu(z)$ was defined in (10.5) as the number of parents of node z and C_i denotes the worst case transition probability for kernel Q_i, as in Eq. (10.8).

For notation simplicity the product in (10.12) is over all patches $z \in \mathbf{A}$, but the corresponding C_is are 1 for those z that are not ancestors of x in the propagation graph.

The result from Theorem 10.1 reflects some of the intuitive ideas that led the design of patch matching algorithms. Due to the propagation, all ancestors of x contribute to the probability of x of improving its energy. But not all nodes contribute the same to the bound. The larger $\varepsilon_{z,x}$ the better, as the $C_i(z, .)$ are non-increasing functions (Lemma 10.1). It is therefore important to design the propagation graph and its actions to maximize the $\varepsilon_{z,x}$. The shift propagation actions introduced in [BSFG09a] in the patch matching application can be interpreted under the light of Theorem 10.1 as an heuristic to maximize the levels $\varepsilon_{z,x}$.

Indeed, $\varepsilon_{z,x}$ is given by the minimum energy U_z in the intersection of the sets $A_{z,y}^{-1}(\{U_y \geqslant \varepsilon_{y,x}\})$ for y in the set of nodes to which z propagates. Say y is the right neighbor of z. Then the

propagation action from z to y is a right shift $A_{z,y} = \text{right}(\cdot)$. Suppose that the patches are of size $s \times s$ and the matching cost is a function of the sum of pixel-wise matching errors over the patch (such as any L_p norm). Since the patches at z and y overlap, the energies $U_y(\eta)$ and $U_z(A_{z,y}^{-1}\eta)$ have $s(s-1)$ terms in common, and can expected to be similar. Therefore the minimum value of U_z in $A_{z,y}^{-1}(\{U_y \geqslant \varepsilon_{y,x}\})$ should not be much lower than $\varepsilon_{y,x}$. By assuming certain regularity conditions on the image it is possible to bound the difference between $\varepsilon_{z,x}$ and $\varepsilon_{y,x}$. This is out of the scope of this chapter.

A result that is found in practice is that the rate at which the energy decreases becomes slower as the matches improve. Less contributions from other nodes are taken into account and mostly the random search improves the matching. This is in agreement with the theory, since as ε decrease, the sizes of the allowed set increase and the $\varepsilon_{z,x}$ decrease as well. According to Theorem 10.1, this results in a slower energy decrease.

We can also note that adding more edges to the propagation graph can only improve the convergence rate, requiring a smaller number of iterations to achieve a desired precision in the result (in probability). However, having more edges implies more computation each iteration. The optimal number of propagation links results from a trade-off between the computational effort per iteration and its impact on reducing the energy.

Most variants of PatchMatch use the same propagation graph and change the random search steps (including the initialization). This can have a dramatic effect of the convergence. In our bound the choice of the random search determines the coefficients C_1 and C_2 which depend directly on the search transition kernel. Two examples of random searches will be reviewed in §10.4.

In the case of the single nearest neighbor ($k = 1$) the bound from Theorem 10.1 can be improved. The improvement comes from a better version of the C_i coefficients that considers the transition probability of $\text{merge}_x^k(\eta \cup S_i\eta)$. The details are provided in the supplementary material.

10.3.2 Uniform decay and convergence in the mean

One of the advantages of *PatchMatch* algorithms is that the NNF converges rapidly as a whole. We now give bounds on uniform convergence and convergence in the mean.

Theorem 10.2. *Consider the field of candidate matches at iteration n, φ^n. Define φ^{n+1} by applying an iteration of the Generic PatchMatch in Algorithm 28. Then, for all $\varepsilon > 0$ we have*

$$\mathbb{P}(\|U_\cdot(\varphi_\cdot^{n+1})\|_\infty \geqslant \varepsilon) \leqslant \mathbb{P}(\|U_\cdot(\varphi_\cdot^{n})\|_\infty \geqslant \varepsilon) \prod_{z \in \mathbf{A}} \Big(C_2(z, \{U_z \geqslant \varepsilon\})^{\mu(z)} C_1(z, \{U_z \geqslant \varepsilon\}) \Big). \quad (10.13)$$

Theorems 10.1 and 10.2, together with the assumption (10.7) on the transition kernels Q_i, imply the convergence in probability of *PatchMatch* algorithms. Assumption (10.7) is necessary to ensure that $C_i(z, \varepsilon) < 1$ for $\varepsilon > 0$. A stronger convergence can also be shown once we considered transition kernels Q_i having this good property, the following result shows convergence in the mean, both point-wise and uniformly for the whole energy field.

Corollary 10.1. *Assume that for any pair (η, ξ) of sets of k candidate matches $Q_1(\eta, \xi) > 0$ (or $Q_2(\eta, \xi) > 0$). Let (φ^n) be a sequence defined by Algorithm 28. Then $\forall x \in \boldsymbol{A}, \mathbb{E}[U_x(\varphi_x^n)] \xrightarrow[n\to\infty]{} 0$ and $\mathbb{E}[\|U_\cdot(\varphi_\cdot^n)\|_\infty] \xrightarrow[n\to\infty]{} 0$.*

This assumption on Q_i is reasonable when the universe of candidates is finite, such as when searching matching patches in a database image $\mathbf{B}$. Some variants of *PatchMatch* have been

used to minimize a field of functions over continuous parameters (scales and rotations [BSGF10], planes in 3D [BRR11]). The previous corollary does not apply in these cases. Nevertheless both Theorems 10.1 and 10.2 remain valid even without assumption (10.7).

10.4 Specific *PatchMatch* algorithms

We now derive more specific bounds for two particular cases of the generic *PatchMatch* algorithm found in the literature.

10.4.1 The original *PatchMatch* algorithm

The original *PatchMatch* algorithm was introduced in [BSFG09a] for $k = 1$, and then generalized to k-nearest neighbors in [BSGF10] (the *heap algorithm*) together with other generalizations. Most of them are covered by Theorem 10.1 as they are particular cases of Algorithm 28. In this section, we derive a more specific bound for one of these variants.

The field φ of k-matches is initialized at random, by uniform sampling from image $\mathbf{B}$. The propagation graph is the basic one presented in §10.2.2, with the shift actions. For the forward propagation these are $A(x, \text{right}(x)) = \text{right}(\cdot)$, $A(x, \text{down}(x)) = \text{down}(\cdot)$.

We define S_1 by taking samples around the current best candidate (the *RS best* variant in [BSGF10]). In [BSGF10] these samples are drawn from a sequence of transition probabilities. The qth sample is sampled uniformly from a box of size $[-d\alpha^q, d\alpha^q]^2$ centered at the current match, for $q = 0, 1, \ldots, q_{\max}$, $\alpha = 0.5$ and $d = \max\{W, H\}$ for a database image $\mathbf{B}$ of size $W \times H$. The number of samples is chosen so that the smallest box is larger than a pixel. The first sample is taken uniformly on the whole image $\mathbf{B}$, so that there is a positive probability of transitioning from and to any two patches in $\mathbf{B}$. For simplicity, we assume that the random search operator $S_1\varphi_x^n$ draws k independent patches in $\mathbf{B}$ with the same probability distribution $Q'(\tilde{\varphi}_x, .)$, where $\tilde{\varphi}_x$ is the best current match. For all $\phi \in \mathbf{B}$, the support of $Q'(\phi, \cdot)$ covers all patches in $\mathbf{B}$, so as in [BSFG09a, BSGF10], there is always a positive probability of transitioning between any two patches in $\mathbf{B}$.

Without loss of generality, to simplify the notation we consider that the propagation graph is the same for all iterations (i.e. no alternation between iterations).

Proposition 10.1. *The specific basic PatchMatch algorithm described above converges in probability to a NNF which minimizes the energy, namely*

$$\lim_{n\to\infty} \mathbb{P}(U_x(\varphi_x^n) \geqslant \varepsilon) = 0, \forall \varepsilon > 0, x \in \boldsymbol{A}, \tag{10.14}$$

with a geometric convergence rate. Moreover for all $\varepsilon > 0$, for all $x \in \boldsymbol{A}$, we have that

$$\mathbb{P}(U_x(\varphi_x^{n+1}) \geqslant \varepsilon) \leqslant \mathbb{P}(U_x(\varphi_x^n) \geqslant \varepsilon) \prod_{z \in \mathbf{A}} \left(1 - (1 - C'(z, \varepsilon_{z,x}))^k\right),$$

with $C'(z, \alpha) := \sup_\eta Q'(\eta, \{U_z \geqslant a\})$. For $\alpha > 0$ we can guarantee that $C'(z, \alpha) < 1$.

Corollary 10.2. *If $k = 1$ the upper bound can be written as*

$$\mathbb{P}(U_x(\varphi_x^{n+1}) \geqslant \varepsilon) \leqslant \prod_{z \in \boldsymbol{A}} C'(z, \varepsilon_{z,x}) \mathbb{P}(U_x(\varphi_x^n) \geqslant \varepsilon).$$

with $C'(z, \alpha) := \sup_{\eta \in \{U_z \geqslant a\}} Q'(\eta, \{U_z \geqslant a\})$.

Corollary 10.2 allows a direct comparison of our bound with the bound derived in [ACF12]. Both bounds have the same structure. However, the energy levels ε_{zx} [ACF12] are smaller than ours causing a looser bound.

10.4.2 The CSH algorithm

Coherency Sensitive Hashing (CSH) was introduced in [KA11]. It uses Locality Sensitive Hashing (LSH) [IM98] to improve the random search and to add propagation edges.

LSH is a method for nearest neighbors search based on partitioning the search space using a series of hash functions h drawn randomly from a family $\mathcal{H}$. Each hash function partitions the space in "bins" containing points with the same hash value. The family $\mathcal{H}$ has the property that nearby points collide in the same bin with high probability, while far away points do so with smaller probability. This is made precise by the following definition 10.6.

Definition 10.6. *A family of functions $\mathcal{H} = \{h : \mathbb{R}^d \to U\}$, where U is the set of hashes, is called (R, cR, p_1, p_2)-sensitive if for any $p, q \in \mathbb{R}^d$ (for simplicity of notation, we write $\mathbb{P}_{\mathcal{H}}(\,.\,) = \mathbb{P}(\,.\mid \mathcal{H})$)*

(1) if $\|p - q\| \leqslant R$ then $\mathbb{P}_{\mathcal{H}}(h(q) = h(p)) \geqslant p_1$,

(2) if $\|p - q\| \geqslant cR$ then $\mathbb{P}_{\mathcal{H}}(h(q) = h(p)) \leqslant p_2$.

This is useful for nearest neighbors search when $p_1 > p_2$.

Instead of using directly the elements from $\mathcal{H}$, a second family of functions $\mathcal{G}$, called an OR family, is created. The function $g \in \mathcal{G}$ is based on a set of n random functions $h_1, \ldots, h_n$ from $\mathcal{H}$ such that for all p, q, $g(p) = g(q)$ if and only if there exist $i \in [\![1, n]\!]$ such that $h_i(p) = h_i(q)$. This will be the set of functions used to define the algorithm.

Lemma 10.3. *If $\mathcal{H}$ is (R, cR, p_1, p_2)-sensitive then an OR family $\mathcal{G}$ created using n functions from $\mathcal{H}$ is $(R, cR, 1 - (1 - p_1)^n, 1 - (1 - p_2)^n)$-sensitive.*

In CSH, hash functions drawn randomly at each iteration are used both to define the random search operator S_1 and the randomized propagation operator S_2. Let us define the projection bin of a patch $p \in \mathbb{R}^d$ as

$$\mathcal{B}_g(p) = \{q \in \mathbf{B} \mid g(p) = g(q)\}. \tag{10.15}$$

These projection bins depend on the random choice of the hash function, and are used to define the candidate set sampling operators. The random search S_1 is simply the projection bin of the query patch x, i.e. $S_1\varphi_x := \mathcal{B}_g(x)$. The randomized propagation is defined as the union of the projection bins corresponding to the best b propagated candidates, where $b \leqslant k$ is a parameter of the method, that is:

$$S_2 A_{y,x}\varphi_y = \cup_{l=1}^{b} \mathcal{B}_g([A_{y,x}\varphi_y]_l), \quad y \sim x. \tag{10.16}$$

Here $[A_{y,x}\varphi_y]_l$, $l = 1, \ldots, k$ are the propagated candidates.

Note that $S_1\varphi_x$ does not depend on the candidate list, but only on the query patch x. This particularity, together with the properties of the hashing family $\mathcal{H}$, can be exploited to derive a tighter upper bound than the one given in Theorem 10.1. CSH also uses hashing over the query image $\mathbf{A}$ to connect similar patches in the propagation graph. Our result is valid for any propagation graph.

Proposition 10.2. *For a (R, cR, p_1, p_2)-sensitive family of hashing functions such that $R \geqslant \max_{z \in \boldsymbol{A}} K_z$ (see Definition 10.1), the sequence (φ^n) defined by the CSH algorithm converges in probability to a minimizer of the total energy,*

$$\lim_{n \to \infty} \mathbb{P}(U_x(\varphi_x^n) \geqslant \varepsilon) = 0, \forall \varepsilon > 0, x \in \boldsymbol{A}, \tag{10.17}$$

with a geometric convergence rate. Moreover for all $\varepsilon > 0$, for all $x \in \boldsymbol{A}$,

$$\mathbb{P}(U_x(\varphi_x^{n+1}) \geqslant \varepsilon) \leqslant \mathbb{P}(U_x(\varphi_x^n) \geqslant \varepsilon) \tag{10.18}$$

$$\prod_{z \in \mathbf{A}} C_2(z, \ell^{\varepsilon}_{z,x})^{\mu(z)} f(p_1, \varepsilon_{z,x}), \tag{10.19}$$

where $f(\alpha, \beta) = (1 - \alpha^k)$ if $\beta > 0$, 1 otherwise.

In practice it is possible to compute an (R, cR, p_1, p_2)-family of hash functions such that p_1 is large enough to outperform the transition kernels used by the original *PatchMatch* algorithm. This explains the improved convergence rate of CSH reported in [KA11].

10.5 Experiments and discussion

We show experiments comparing the convergence predicted by the theory with the one found in practice, for the case of the original PatchMatch algorithm with $k = 1$ [BSFG09a]. The code to reproduce the experiments is available at `https://github.com/pariasm/pm-bound`.

10.5.1 Computation of theoretical bound

Given an energy value ε and a patch x in the query image $\mathbf{A}$, the computation of the bound requires computing the energy levels $\varepsilon_{z,x}$ and the worst case transition probability $C_1(z, \varepsilon_{z,x})$ for all ancestors z of x. The ancestors are all pixels located above and to the left of x during the forward propagation, and below and on the right of x during the backward pass. The original PatchMatch algorithm does not consider a randomized propagation $S_2 A_{z,y} \varphi_y^{n+1}$, thus in the following we will drop the subindex for C_1 and Q_1.

The computation of the levels $\varepsilon_{z,x}$ is straightforward. We loop on the ancestors starting from x. Following the inverse propagation ordering we apply the recursion (10.10). The inverse actions $A_{z,y}^{-1}$ are one pixel shifts in the opposite direction to the propagation. For instance if z propagates to y at his right, then $A_{z,y}^{-1}\{U_y \geqslant \varepsilon_{y,x}\}$ results from shifting to the left all elements of $\{U_y \geqslant \varepsilon_{y,x}\} \subseteq \mathbf{B}$.

Figure 10.1 shows examples of the resulting $\varepsilon_{z,x}$ at two image locations and for $\varepsilon = 1$ and $\varepsilon = 15$. The larger the $\varepsilon_{z,x}$ the smaller the coefficient $C(z, \varepsilon_{z,x})$ and the faster the convergence. We can see that the levels $\varepsilon_{z,x}$ at different points x can be very different. Moreover, increasing ε increases the levels $\varepsilon_{z,x}$ locally around x.

To compute the worst case transition probability $C(z, \varepsilon_{z,x})$, we compute for all $\eta \in \{U_z \geqslant \varepsilon_{z,x}\}$ the probability of drawing a sample from the upper-level set. As in the original PatchMatch, we take n independent samples following a sequence of uniform distributions $S_i \eta \sim Q_i(\eta, \cdot)$ on square boxes centered at η with a decreasing sequence of radii.[1] For the worst case transition, we need the n samples to be in $\{U_z \geqslant \varepsilon_{z,x}\}$, thus:

$$C(z, \varepsilon_{z,x}) = \max_{\eta \in \{U_z \geqslant \varepsilon_{z,x}\}} \prod_{i=1}^{n} Q_i(\eta, \{U_z \geqslant \varepsilon_{z,x}\}). \tag{10.20}$$

For each sampling radius r, the probability $Q_i(\cdot, \{U_z \geqslant \varepsilon_{z,x}\})$ can be computed efficiently via a convolution of the indicator function of the upper-level set with a box kernel of size $2r + 1 \times 2r + 1$[2]. We use an integral image implementation to speed-up the computation.

[1] This scheme differs from the one considered in Proposition 10.1, which takes k equally distributed samples $S\eta \sim Q'(\eta, \cdot)$. But the theory applies the same as long as C is in accordance with the sampling procedure.

[2] The kernel needs to be normalized by the area of the portion of the kernel that fits in the image domain.

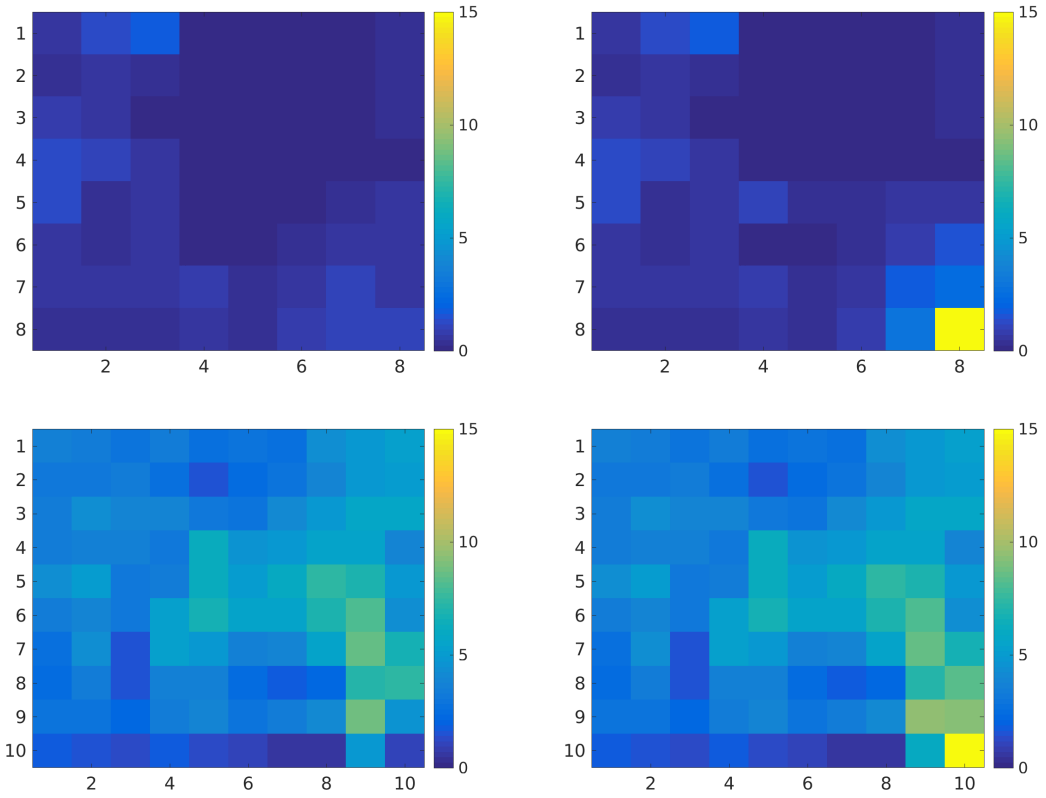


Figure 10.1: We represent here the $\varepsilon_{z,x}$s for two different points x (top and bottom rows) and for $\varepsilon = 1$ (left) and $\varepsilon = 15$ (right). The point x is the bottom right pixel.

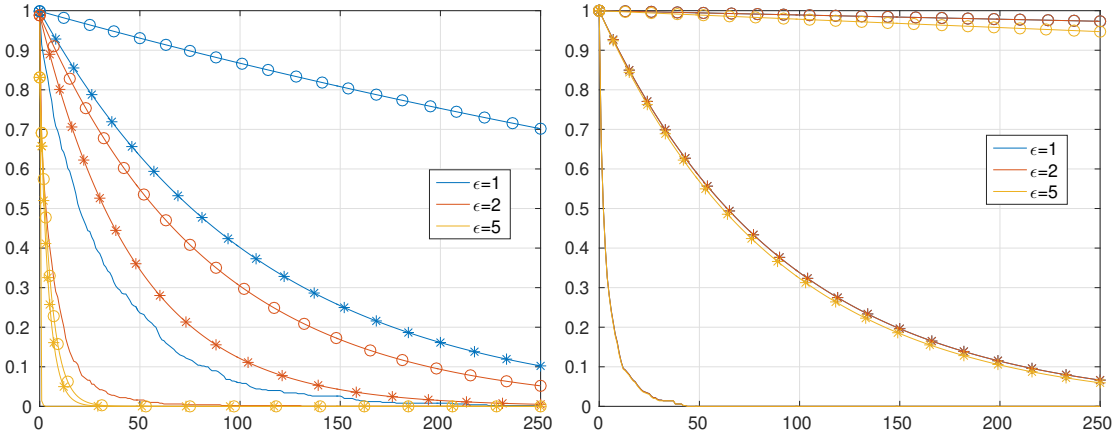


Figure 10.2: The different theoretical bounds are compared to the empirical one at the position of the two points from Figure 10.1 for different values of ε. The plots show the probability than the energy is above different energy levels. The line without marker corresponds to the empirical probability. The line marked with a circle corresponds to the bound presented in [ACF12]. The line marked with a star correspond to the bound presented in this chapter.

10.5.2 Experimental validation

To validate the theory we contrast the predicted evolution of the matching energy with the empirical evolution. Since PatchMatch is a randomized algorithm, we estimate for a given energy level ε the probability that the energy is above ε for each iteration. The empirical probability is computed by running N iterations of PatchMatch a number of trials M. For a patch $x \in \mathbf{A}$ we define $U_x^{n,m}$ the matching energy at iteration n of trial m. Then the empirical probability of $U_x \geqslant \varepsilon$ is estimated as

$$\widehat{P}(x, \varepsilon, n) = \widehat{\mathbb{P}}(U_x(\varphi_x^n) \geqslant \varepsilon) = \frac{1}{M} \sum_{m=1}^{M} \mathbf{1}(U_x^{n,m} \geqslant \varepsilon).$$

The theoretical bound on this probability is given by

$$B(x, \varepsilon, n) = \mathbb{P}(U_x(\varphi_x^0) \geqslant \varepsilon) \cdot K(x, \varepsilon)^n, \tag{10.21}$$

where $K(x, \varepsilon) = \prod_{z \in \mathbf{A}} C(z, \varepsilon_{z,x})$. The candidates in the original PatchMatch are initialized by sampling uniformly over the database image $\mathbf{B}$, thus the initial probabilities can be easily computed as the area ratio between the upper-level set $\{U_x \geqslant \varepsilon\}$ and the image domain $\mathbf{B}$.

In Figure 10.2 we plot the estimated probabilities $\widehat{P}(x, \varepsilon, n)$ together with the theoretical bound $B(x, \varepsilon, n)$ for a patch $x \in \mathbf{A}$ and several values of ε. As expected, the estimated probability is below the worst case bound. For smaller ε the bound becomes looser. For example: for the energy to be below than $\varepsilon = 1$ with a probability of 0.2, the theoretical bound predicts 178 iterations but in practice 60 were needed.

For $k = 1$, the gap between the theoretical bound and the empirical decay is mainly due to upper-bounding the transition probability (10.11) by the worst case probability C. To verify this we use a simplified random search: the sample $S\eta$ is taken uniformly over $\mathbf{B}$ (as in the initialization). In this way, we eliminate the dependence between the sample $S\eta$ and the current candidate η, and we have that $\mathbb{P}(S\varphi_z^n \in \{U_z \geqslant l\} \mid \varphi_z^n \in \{U_z \geqslant l\})$ is the area ratio between the upper level set and the domain of image $\mathbf{B}$, regardless of φ_z^n.

We generate two images of random noise. The query image $\mathbf{A}$ is of size 24×24. It contains $q = 20 \times 20 = 400$ patches of size 5×5. The database image $\mathbf{B}$ is of size 104×104, thus

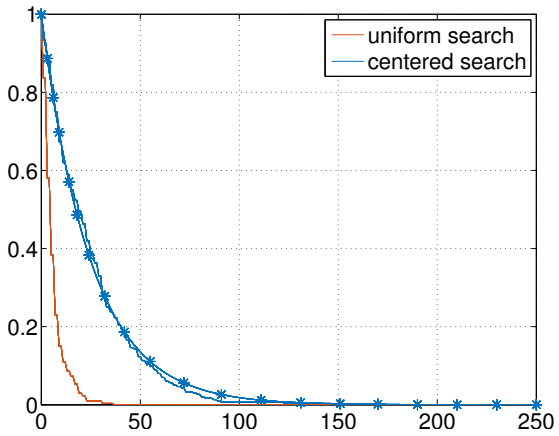


Figure 10.3: The effect of the random search. If we simplify the random search and make it uniform, then convergence is slower and the bound is tight. In this experiment, we match two random noise images. See text for details.

containing $p = 100 \times 100 = 10^4$ patches. As before, we compute the empirical probabilities $\widehat{P}$ and the bound for the bottom right patch. We show two results: one using only the uniform random search, and another one using the original centered random search. The plots correspond to an energy level of $\varepsilon = 0.5$, and only the unique global minimum is below ε. When using the uniform search, the empirical decay matches the theoretical prediction. The centered search shows a faster convergence.

This experiment also shows that the theoretical bound captures the main intuition behind the design of the PatchMatch algorithm: if a region of image **B** is an exact copy of image **A** (or of a region in image **A**) it becomes very likely to find the copied region in the database image. Due to the propagation, as soon as one node in **A** finds its match, it will be propagated to all other nodes. In this case, one can compute exactly the probability that all the ancestors of x miss the copied region. The probability of missing the global optimum is $1 - 1/p$. The probability that all q ancestors of x miss the optimum is therefore $(1 - 1/p)^q$. This coincides with the theoretical bound.

As a final experiment we show results obtained computing C as an "average" transition probability instead of the worst case. This average transition results from assuming in (10.11) that φ_x^n is distributed uniformly over the upper-level set. Figure 10.4 compares the predictions of this average C with the worst case C. While we do not have theoretical guaranties on the average C, its behavior that is much closer to the empirical case.

10.6 Conclusions

We presented a theoretical analysis of the convergence of PatchMatch algorithms. For an energy level ε, we show that the probability of having an energy above ε converges to 0 with the number of iterations, and we provide worst case bounds on the convergence rate. Our analysis applies to the case of k nearest neighbors, and to most variants of PatchMatch proposed in the literature. We give specific bounds for two of these algorithms: the original PatchMatch [BSFG09a, BSGF10] and CHS [KA11]. For the case of the original PatchMatch (with $k = 1$) we validate our results by comparing the predicted convergence rate with the one found in practice. The setting of our framework is rather general: the task of patch matching is viewed as an optimization problem where the goal is to minimize several non-convex energies U_x over the same domain. This might allow the application of these techniques to similar optimizations problems in other areas. For

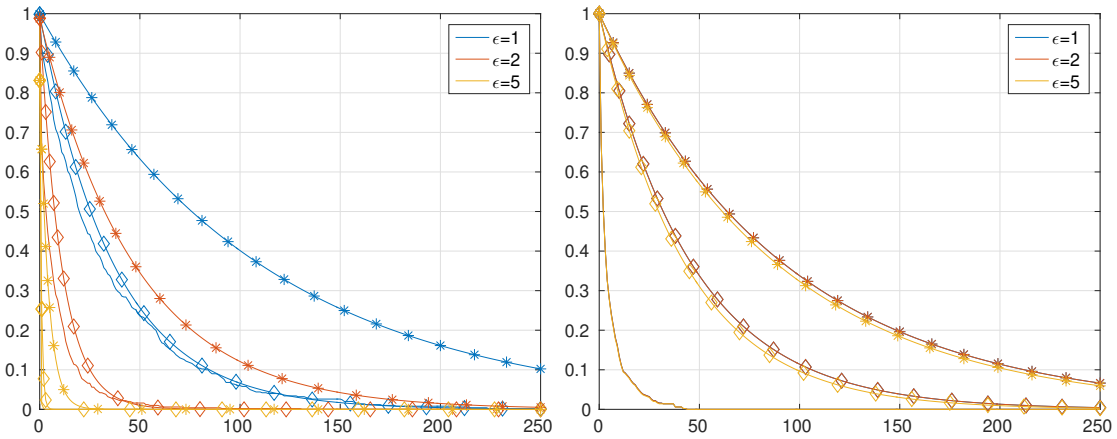


Figure 10.4: Comparison between the "average" and worst-case C. The curve with the diamonds corresponds to the "average" C whereas the stars show the worst case C.

the case of matching patches, it would interesting to precise the link between the regularity of the images and the convergence rate, at least for certain simple models of images.

11 Detection of copy-paste forgeries based on PatchMatch

This chapter presents an implementation and discussion of the recently proposed 'Efficient Dense-Field Copy-Move Forgery Detection' by Cozzolino et al.. This method is a forgery detection based on a dense field of descriptors chosen to be invariant by rotation. Zernike moments were suggested in the original article. An efficient matching of the descriptors is then performed using PatchMatch, which is extremely efficient to find duplicate regions. Regions matched by PatchMatch are processed to find the final detections. This allows a precise and accurate detection of copy-move forgeries inside a single suspicious image. We also extend successfully the method to the use of dense SIFT descriptors and show that they are better at detecting forgeries using Poisson editing.

11.1 Introduction

Copy-move forgeries correspond to the case where a region is copied from the image and then pasted in another position of the same image. The copy could be modified in the meantime, but the detection presented here is only guaranteed to work when a rotation has been applied. Nevertheless we shall see that this method is also mildly robust to other perturbations such as noise. This type of forgery, which can easily be performed with most image editing software tools, is usually used to hide undesired details in the image, or to add new details. Detecting such modifications can be challenging, especially when carried out by a professional image editor.

Many different methods have been developed to detect such modifications. In [Far09], Farid presented a large number of such methods: from the simplest ones such as pixel-based methods to much more complex physics-based methods by which, for example, the illumination of the scene is studied. In this study, we focus on a single family of detection methods. These methods, specialized im detecting copy-move forgeries, are organized in the following way. First, descriptors are computed on the suspicious image. These descriptors are chosen to represent well the local behavior of the suspicious image. They are then matched to each other during a matching step to detect abnormal similarities within the image. In the final step, post-processing is applied to refine detections and diminish the number of false positives. The first papers on this subject proposed classic sparse image descriptors such as SIFT [PL10] or SURF [SB11]. A recent benchmark [CRJ+12] of many different methods based on both sparse descriptors and dense descriptors shows that dense descriptors largely outperform sparse descriptors. For this reason [CPV15] chose to use dense descriptors. Matching the descriptors, or equivalently finding the nearest neighbor for each descriptor, is a well studied problem but actually quite hard. Many different

approaches have been developed but all actually compute approximate matches. The approximation can be computed using partition trees such as the well known KD-trees [Ben75], or VP-trees [Yia93], or adapted hashing functions such as LSH [IM98,GIM+99]. A recent breakthrough for image specific algorithms is *PatchMatch*, developed by Barnes et al. [BSFG09a, BSGF10]. This algorithm relies on the structural properties of the image and multiple sampling, which in practice performs extremely well on dense descriptors. It is also the matching algorithm adopted by the reviewed method.

The rest of the chapter is organized as follows: the descriptors used for the processing, Zernike moments, are presented in Section 11.2. The matching algorithm *PatchMatch* and the modifications implemented to improve the matching for our problem are presented in Section 11.3. Several pre- and post-processing steps presented in Section 11.4 are then applied to obtain precise and accurate detections. Finally, a couple of experiments where we show success and limits of the method are presented in Section 11.5, where we successfully extend the method to dense SIFT descriptors..

11.2 Zernike Moments

The absolute values of Zernike moments are rotation invariant descriptors that are adapted to the task of detecting image repetitions up to a rotation. We address in this section their efficient computation by the Xin et al. [XPL07] method. For a patch u of an image I (defined for now as continuous on $[-1,1]^2$), the Zernike A_{nm} moment of order n and angular dependence m, for $m < n$ and $m \equiv n[2]$, is defined by

$$A_{nm} = \frac{n+1}{\pi} \int\int_{\mathbb{D}} u(x,y)\overline{V_{nm}}(x,y)dxdy, \tag{11.1}$$

where V_{nm} is the complex Zernike polynomial of order n and angular dependence and $A_{nm} = 0$ otherwise (therefore in the following study, the case $m \not\equiv n[2]$ will be omitted). The polynomials are defined over the unit disk $\mathbb{D}(0,1)$ and usually expressed in polar coordinates ρ and θ. They are fully defined by

$$V_{nm}(\rho,\theta) = R_{nm}(\rho)e^{im\theta}, \tag{11.2}$$

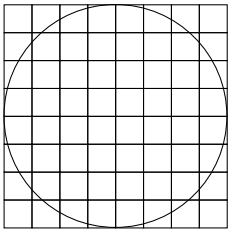
$$R_{nm}(\rho) = \sum_{s=0}^{\lfloor \frac{n-m}{2} \rfloor} \frac{(-1)^s (n-s)! \rho^{n-2s}}{s!(\lfloor \frac{n+m}{2} \rfloor - s)!(\lfloor \frac{n-m}{2} \rfloor - s)!}. \tag{11.3}$$

Zernike moments can be interpreted as scalar products between the image u and the Zernike polynomials. The absolute value of these moments is rotationally invariant. Indeed, consider a rotation of angle α of the given image, or equivalently, a rotation of angle α of the Zernike polynomials

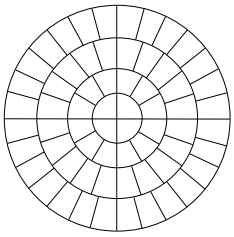
$$V_{nm}^{\alpha} = V_{nm}e^{i\alpha},$$

the studied moment A_{nm}^{α} would then be

$$\begin{aligned}
A_{nm}^{\alpha} &= \frac{n+1}{\pi} \int\int_{\mathbb{D}} u(x,y)\overline{V_{nm}^{\alpha}}(x,y)dxdy && (11.4)\\
&= \frac{n+1}{\pi} \int\int_{\mathbb{D}} u(x,y)\overline{V_{nm}}(x,y)e^{-i\alpha}dxdy && (11.5)\\
&= \frac{n+1}{\pi} \left(\int\int_{\mathbb{D}} u(x,y)\overline{V_{nm}}(x,y)dxdy \right) e^{-i\alpha} && (11.6)\\
&= A_{nm}e^{-i\alpha}, && (11.7)
\end{aligned}$$



(a) Cartesian sampling of the unit disk



(b) Polar sampling of the unit disk

Figure 11.1: Different types of samplings for the Zernike moments.

and therefore

$$|A_{nm}^{\alpha}| = |A_{nm}|, \tag{11.8}$$

which confirms the rotation invariance of the magnitude of these moments.

In our case, the moments will be computed on each patch of the image (considered as a small image itself). Within the same framework presented previously, one can also define translation invariant descriptors by computing a dense field of descriptors over the entire image. The computation is trickier than it appears at first sight because the image (noted u and so far considered continuous) is sampled over a Cartesian grid whereas Zernike polynomials are more naturally expressed in polar coordinates. The sampled version of I and u with a sampling step $\frac{1}{sp}$ will be noted respectively $\tilde{I}$ and $\tilde{u}$. The coordinates (x_0, y_0) denote the center of $\tilde{u}$ in $\tilde{I}$ and $\tilde{u}$ is defined on $[\![-sp, sp]\!]$. The easiest solution would be to estimate the integral from Equation (11.1) by sampling over the Cartesian grid by

$$A_{nm} = \sum_{s_x=-sp}^{sp} \sum_{s_y=-sp}^{sp} \tilde{u}(s_x, s_y)\overline{V_{nm}}(\sqrt{s_x^2 + s_y^2}, atan2(s_y, s_x)). \tag{11.9}$$

Xin et al. have shown in [XPL07] that this approximation suffers from multiple drawbacks and is not properly rotation invariant. The rotational invariance being one of the most important features of the Zernike moments in our case, we shall use the more accurate estimation proposed in [XPL07]. First, we need to resample the image. Indeed the original Cartesian sampling is not adapted to the unit disk. Figure 11.1a shows that whatever the choice of pixels taken to do the integration on, it will never truly form a disk; therefore leading to approximation in the computations. An adequate image partitioning is required to care for the rotation invariance. First, the unit disk is split into K regions along the radial direction. These regions are then divided into "pixels" with the same area. For this reason each region is split into $(2k+1)L$ subregions. This sampling of the unit disk is presented in Figure 11.1b. It has the nice property of not losing resolution compared to the original image. In addition, the new pixels have approximately the same area as the old ones. The image is considered piecewise constant over these "pixels". The value over such a pixel, noted $f(\rho_{kl}, \theta_{kl})$, is computed by image interpolation. Xin et al. recommended the usage of a bicubic interpolation. This yields

$$f(\rho_k, \theta_{kl}) = \sum_{s_x=\lfloor \rho_k \cos(\theta_{kl}) \rfloor - 1}^{\lfloor \rho_k \cos(\theta_{kl}) \rfloor + 2} \sum_{s_y=\lfloor \rho_k \sin(\theta_{kl}) \rfloor - 1}^{\lfloor \rho_k \sin(\theta_{kl}) \rfloor + 2} \tilde{u}(s_x, s_y) h\left(\rho_k \cos(\theta_{kl}) - s_x\right) h\left(\rho_k \sin(\theta_{kl}) - s_y\right), \tag{11.10}$$

where

$$h(x) = \begin{cases} \frac{3}{2}|x|^3 - \frac{5}{2}|x|^2 + 1 & \text{if } |x| \leq 1 \\ -\frac{1}{2}|x|^3 + \frac{5}{2}|x|^2 - 4|x| + 2 & \text{if } 1 < |x| \leq 2 \\ 0 & \text{otherwise} \end{cases}. \tag{11.11}$$

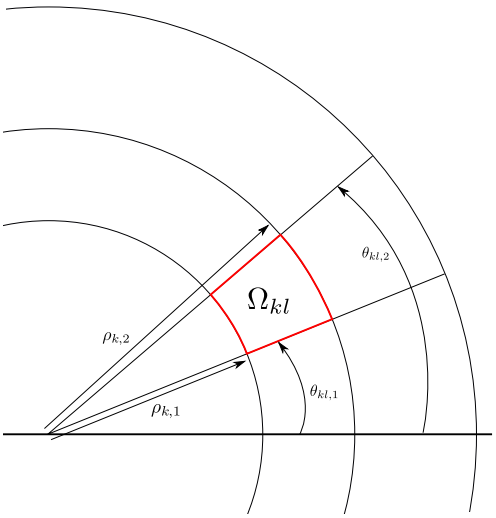


Figure 11.2: Notations for Equations (11.15), (11.16) and (11.17).

We can now express Zernike moments in polar coordinates by a simple change of variable,

$$A_{nm} = \frac{n+1}{\pi} \int_0^{2\pi} \int_0^1 u(\rho, \theta) \overline{V_{nm}}(\rho, \theta) \rho d\rho d\theta. \tag{11.12}$$

This leads to the following estimate of A_{nm} by sampling the integral over the newly created regions

$$A_{nm} = \frac{n+1}{\pi} \sum_k \sum_l f(\rho_k, \theta_{kl}) w_{nm}(\rho_k, \theta_{kl}), \tag{11.13}$$

where w_{nm} corresponds to

$$w_{nm}(\rho_k, \theta_{kl}) = \int \int_{\Omega_{kl}} R_{nm}(\rho) e^{-im\theta} \rho d\rho d\theta \tag{11.14}$$

$$= \left(\int_{\rho_{k,1}}^{\rho_{k,2}} R_{nm}(\rho) \rho d\rho \right) \left(\int_{\theta_{kl,1}}^{\theta_{kl,2}} e^{-im\theta} d\theta \right), \tag{11.15}$$

where the support of the pixel Ω_{kl} is parameterized by $\rho_{k,1}, \rho_{k,2}, \theta_{kl,1}, \theta_{kl,2}$ (see Figure 11.2). Besides the two previous integrals can be computed in closed form

$$\int_{\rho_{k,1}}^{\rho_{k,2}} R_{nm}(\rho) \rho d\rho = \left(\sum_{s=0}^{\lfloor \frac{n-m}{2} \rfloor} \frac{(-1)^s (n-s)! \left(\rho_{k,2}^{n-2s+2} - \rho_{k,1}^{n-2s+2} \right)}{(n-2s+2) s! (\lfloor \frac{n+m}{2} \rfloor - s)! (\lfloor \frac{n-m}{2} \rfloor - s)!} \right), \tag{11.16}$$

and

$$\int_{\theta_{kl,1}}^{\theta_{kl,2}} e^{-im\theta} d\theta = \begin{cases} \theta_{kl,2} - \theta_{kl,1} & \text{if } m = 0 \\ \frac{i}{m} \left(e^{-im\theta_{kl,2}} - e^{-im\theta_{kl,1}} \right) & \text{otherwise} \end{cases}. \tag{11.17}$$

Xin et al. have shown that computing the moments using this framework allows for a much better rotation invariance compared to the regular Cartesian approximation. The sums in Equation (11.13) can be reordered to compute efficiently the entire set of Zernike moments for the

entire image $\tilde{I}$. Indeed A_{nm} can be rewritten

$$\begin{aligned}
A_{nm} &= \frac{n+1}{\pi} \sum_k \sum_l f(\rho_k, \theta_{kl}) w_{nm}(\rho_k, \theta_{kl}) \\
&= \frac{n+1}{\pi} \sum_k \sum_l \sum_{s_x} \sum_{s_y} \tilde{u}(s_x, s_y) h\left(\rho_k \cos(\theta_{kl}) - s_x\right) h\left(\rho_k \sin(\theta_{kl}) - s_y\right) w_{nm}(\rho_k, \theta_{kl}) \\
&= \frac{n+1}{\pi} \sum_{s_x} \sum_{s_y} \sum_k \sum_l \tilde{u}(s_x, s_y) h\left(\rho_k \cos(\theta_{kl}) - s_x\right) h\left(\rho_k \sin(\theta_{kl}) - s_y\right) w_{nm}(\rho_k, \theta_{kl}) \\
&= \frac{n+1}{\pi} \sum_{s_x} \sum_{s_y} \tilde{u}(s_x, s_y) \left(\sum_k \sum_l h\left(\rho_k \cos(\theta_{kl}) - s_x\right) h\left(\rho_k \sin(\theta_{kl}) - s_y\right) w_{nm}(\rho_k, \theta_{kl}) \right) \\
&= \frac{n+1}{\pi} \sum_{s_x} \sum_{s_y} \tilde{u}(s_x, s_y) F^{nm}_{s_x, s_y}, \qquad (11.18)
\end{aligned}$$

which can now be used to define an "image" of zernike moments such that

$$\begin{aligned}
A_{nm}(x_0, y_0) &= \frac{n+1}{\pi} \sum_{s_x} \sum_{s_y} \tilde{u}(s_x, s_y) F^{nm}_{s_x, s_y} \\
&= \frac{n+1}{\pi} \sum_{s_x} \sum_{s_y} \tilde{I}\left(x_0 + s_x, y_0 + s_y\right) F^{nm}_{s_x, s_y}. \qquad (11.19)
\end{aligned}$$

Equation (11.19) shows that the A_{nm}s can be computed efficiently by convolving the image $\tilde{I}$ with the newly defined filter F^{nm}. A null extension is chosen for the convolution for efficiency and simplicity. Indeed, the moments computed on the border won't have any of the properties required for the matching, whatever the extension (in particular invariance to rotation is the hardest to ensure); therefore they should be discarded anyway in the following steps and the choice should be made only for computation efficiency. The pseudocode for the full computation of the dense field of moments for a given image is given in Algorithm 19. It yields both translation and rotation invariant descriptors with which copy-move forgeries will be detected. The two versions of Zernike moments will be compared for the problem of forgery detection with rotation in Section 11.5.1.

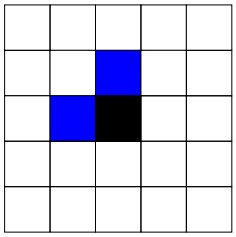
11.3 Patchmatch

In order to detect forgeries, we'll have to find the nearest neighbors of the descriptors from Section 11.2. Usually finding nearest neighbors in high dimension is a rather difficult problem but, for the specific case of images, the *PatchMatch* heuristic is faster and more efficient than non-specific heuristics. Instead of using the original *PatchMatch* presented in Chapter 10, we use an improved version that has a faster convergence. In a follow-up paper [BSGF10], the authors of *PatchMatch* proposed several other versions of the algorithm to either solve slightly different but similar problems or to improve the speed and/or the quality of the matching. In particular, they considered the problem of matching modulo rotations and scaling. While the modification for rotation and scaling yields a slow convergence rate in practice (the space of possible matches becoming far too large for the random search), and is not adapted to rotation invariant descriptors (it explores the entire space of rotations), we will use the modifications suggested when applying *PatchMatch* to the same image. The idea behind this modification is to try to take advantage of the natural symmetric consistency within the matches. Indeed, if $\mathbf{b}$ is the current candidate as nearest neighbor of $\mathbf{a}$, then $\mathbf{a}$ is a good candidate for the nearest neighbor of $\mathbf{b}$.

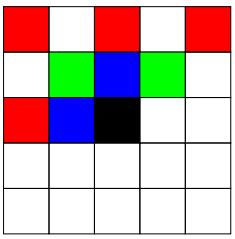
Algorithm 29: Zernike moments computation

input : An image I, a half-size of patch sp, a maximum order o
output : A dense descriptor map Z

foreach (n, m) *s.t.* $n \in [\![1, o]\!], m \in [\![1, n]\!]$ *and* $n \equiv m[2]$ **do**
  for $s = 0$ *to* $\frac{n-m}{2}$ **do**
    $C_{nms} \leftarrow \frac{(-1)^s (n-s)!}{(n-2s+2)s!(\frac{n+m}{2}-s)!(\frac{n-m}{2}-s)!}$ // Compute the coefficients for Equation (11.16)
for $\rho = 0$ *to* $sp - 1$ **do**
  for $\theta = 0$ *to* $4(2\rho + 1) - 1$ **do**
    foreach (n, m) *s.t.* $n \in [\![1, o]\!], m \in [\![1, n]\!]$ *and* $n \equiv m[2]$ **do**
      $w \leftarrow 0$ // Initialize the weights
      for $s = 0$ *to* $\frac{n-m}{2}$ **do**
        $w \leftarrow w + C_{nms} \left(\left(\frac{\rho+1}{sp} \right)^{n-2s+2} - \left(\frac{\rho}{sp} \right)^{n-2s+2} \right)$ // See Equation (11.16)
      if $m = 0$ **then**
        $w \leftarrow w \times \frac{2\pi}{4(2\rho+1)}$ // See Equation (11.17)
      else
        $w \leftarrow w \times \frac{i}{m} \left(e^{-im\frac{2(\theta+1)\pi}{4(2\rho+1)}} - e^{-im\frac{2\theta\pi}{4(2\rho+1)}} \right)$ // See Equation (11.17)
      $F^{nm}_{s_x s_y} \leftarrow 0$
      for $s_x = \left\lfloor \rho \cos\left(\frac{2\theta\pi}{4(2\rho+1)}\right) \right\rfloor - 1$ *to* $\left\lfloor \rho \cos\left(\frac{2\theta\pi}{4(2\rho+1)}\right) \right\rfloor + 2$ **do**
        for $s_y = \left\lfloor \rho \sin\left(\frac{2\theta\pi}{4(2\rho+1)}\right) \right\rfloor - 1$ *to* $\left\lfloor \rho \sin\left(\frac{2\theta\pi}{4(2\rho+1)}\right) \right\rfloor + 2$ **do**
          $F^{nm}_{s_x s_y} \leftarrow h\left(\rho \cos\left(\frac{2\theta\pi}{4(2\rho+1)}\right) - s_x\right) h\left(\rho \sin\left(\frac{2\theta\pi}{4(2\rho+1)}\right) - s_y\right) w + F^{nm}_{s_x s_y}$ See Equation (11.18)
foreach *filter* F^{nm} *computed previously* **do**
  Convolve I (with a null extension) with F^{nm} and store its modulus in ZM_{nm}.



(a) Original *PatchMatch* propagation: elements in blue are propagated to the one in black



(b) Improved *PatchMatch* propagation: elements in blue (original propagations), green (completed zero order propagations) and red (first order propagations) are propagated to the one in black

Figure 11.3: Different types of *PatchMatch* propagations.

This will be combined with the modifications suggested by Cozzolino et al. in [CPV15] to take rotation invariant descriptors into consideration and improve their matching. In order to increase the quality of the matches – when taking rotations into consideration – two more propagations are added. These two propagations correspond to a propagation from the top-left patch as well as from the top-right one. This means that information is propagated to all connecting patches. These propagations are represented in green in Figure 11.3b. A second set of propagations, called first-order propagations, are also added to the list of propagations. These propagations are different from the previous ones because they are not simply a displacement propagation. They propagate linearly varying offset modifications. In order to simplify the notations for some of the equations, we also introduce the optimal displacement $\mathbf{s}$ alongside an ANNF $\mathbf{f}$ such that $\mathbf{f}(\mathbf{a}) = \mathbf{a} + \mathbf{s}(\mathbf{a})$. In the following the ANNF and the optimal displacement will be used interchangeably, the one leading to the simplest equations will be chosen. For two functions f and g, the composition of functions is written $f \circ g = f(g(.))$. Using a rough Taylor expansion, an approximation of a function f can be made

$$\begin{aligned} f(x+1) &\approx f(x) + \frac{df(x)}{dx} \\ &\approx f(x) + \frac{f(x) - f(x-1)}{1} \\ &\approx 2f(x) - f(x-1). \end{aligned} \tag{11.20}$$

This approximation is classic in numerical analysis. Translating Equation (11.20) to the current problem, A being any composition of L, R, U and D, leads to

$$\mathbf{s}(\mathbf{a}) \approx 2\mathbf{s}(A(\mathbf{a})) - \mathbf{s}(A \circ A(\mathbf{a})). \tag{11.21}$$

The first-order nodes used for these types of propagations are represented in red in Figure 11.3b. In the end, a total of eight propagations of the different types is done during every iteration of *PatchMatch*.

Adding all these modifications to the original *PatchMatch* algorithm presented in Chapter 10 gives us the version used to solve the problem at hand. The final modified *PatchMatch* algorithm is summarized in Algorithm 31.

11.4 Forgery Detection

Once the displacement map, a function that for each patch of the reference image, associates its nearest neighbor, has been computed using *PatchMatch*, forgeries can be detected. In order

Algorithm 30: Modified *PatchMatch*

Input: A an image, d a distance function for patches, N a number of iterations, w (usually the width of A) and $\alpha = \frac{1}{2}$ parameters for the random search, k the number of nearest neighbors to be computed, a minimum distance threshold τ_m

Result: $\mathbf{f}$, a k-ANNF from A to A

Initialization:
foreach *patch* $\boldsymbol{a}$ *in* A **do**
  Sample a random patch in A for $\mathbf{f}(\mathbf{a})$
Iterations:
for $n = 1$ *to* N **do**
  foreach *patch* $\boldsymbol{a}$ *in* A *from the top left to the bottom right if* n *is odd, from the bottom right to the top left otherwise* **do**
    Propagation:
    if n *is even* **then**
      For each propagation, reject candidates that are closer than τ_m from $\mathbf{a}$ in the image:
      $\mathbf{f}(\mathbf{a}) \leftarrow \arg\min_{\mathbf{p} \in \{\mathbf{f}(\mathbf{a}), R(\mathbf{f}(L(\mathbf{a})))\}} \{d(\mathbf{p}, \mathbf{a})\}$ `// Zero-order propagation`
      $\mathbf{f}(\mathbf{a}) \leftarrow \arg\min_{\mathbf{p} \in \{\mathbf{f}(\mathbf{a}), D(\mathbf{f}(U(\mathbf{a})))\}} \{d(\mathbf{p}, \mathbf{a})\}$ `// Zero-order propagation`
      $\mathbf{f}(\mathbf{a}) \leftarrow \arg\min_{\mathbf{p} \in \{\mathbf{f}(\mathbf{a}), D \circ L(\mathbf{f}(U \circ R(\mathbf{a})))\}} \{d(\mathbf{p}, \mathbf{a})\}$ `// Zero-order propagation`
      $\mathbf{f}(\mathbf{a}) \leftarrow \arg\min_{\mathbf{p} \in \{\mathbf{f}(\mathbf{a}), D \circ R(\mathbf{f}(U \circ L(\mathbf{a})))\}} \{d(\mathbf{p}, \mathbf{a})\}$ `// Zero-order propagation`
      $\mathbf{f}(\mathbf{a}) \leftarrow \arg\min_{\mathbf{p} \in \{\mathbf{f}(\mathbf{a}), \mathbf{a} + 2\mathbf{s}(L(\mathbf{a})) - \mathbf{s}(L \circ L(\mathbf{a}))\}} \{d(\mathbf{p}, \mathbf{a})\}$ `// First-order propagation`
      $\mathbf{f}(\mathbf{a}) \leftarrow \arg\min_{\mathbf{p} \in \{\mathbf{f}(\mathbf{a}), \mathbf{a} + 2\mathbf{s}(U(\mathbf{a})) - \mathbf{s}(U \circ U(\mathbf{a}))\}} \{d(\mathbf{p}, \mathbf{a})\}$ `// First-order propagation`
      $\mathbf{f}(\mathbf{a}) \leftarrow \arg\min_{\mathbf{p} \in \{\mathbf{f}(\mathbf{a}), \mathbf{a} + 2\mathbf{s}(U \circ L(\mathbf{a})) - \mathbf{s}(U \circ L \circ U \circ L(\mathbf{a}))\}} \{d(\mathbf{p}, \mathbf{a})\}$ `// First-order propagation`
      $\mathbf{f}(\mathbf{a}) \leftarrow \arg\min_{\mathbf{p} \in \{\mathbf{f}(\mathbf{a}), \mathbf{a} + 2\mathbf{s}(U \circ R(\mathbf{a})) - \mathbf{s}(U \circ R \circ U \circ R(\mathbf{a}))\}} \{d(\mathbf{p}, \mathbf{a})\}$ `// First-order propagation`
    else
      Do the reverse propagations
  foreach *patch* $\boldsymbol{a}$ *in* A **do**
    Symmetrization:
    $\mathbf{f}(\mathbf{f}(\mathbf{a})) \leftarrow \arg\min_{\mathbf{p} \in \{\mathbf{f}(\mathbf{f}(\mathbf{a})), \mathbf{a}\}} \{d(\mathbf{p}, \mathbf{f}(\mathbf{a}))\}$
  foreach *patch* $\boldsymbol{a}$ *in* A **do**
    Random search:
    $i \leftarrow 0$
    while $w\alpha^i > 1$ `// Multiple scale random search` **do**
      $r \sim \mathcal{U}([\![-\lfloor w\alpha^i \rfloor; \lfloor w\alpha^i \rfloor]\!] \times [\![-\lfloor w\alpha^i \rfloor; \lfloor w\alpha^i \rfloor]\!])$
      if $\boldsymbol{f}(\boldsymbol{a}) + r$ *is further than* τ_m *from* $\boldsymbol{a}$ **then**
        $\mathbf{f}(\mathbf{a}) \leftarrow \arg\min_{\mathbf{p} \in \{\mathbf{f}(\mathbf{a}), \mathbf{f}(\mathbf{a}) + r\}} \{d(\mathbf{p}, \mathbf{a})\}$
      $i \leftarrow i + 1$

to avoid false alarms, multiple pre-processing and post-processing techniques are applied. The different steps for the detection are:

1. Application of a median filter.
2. Computation of an error map and early detections.
3. Remove detections that are too small.
4. Remove detections that are too close.
5. Symmetrize the detections.
6. Dilate the detections.

The complete algorithm performing forgery detection is presented in Algorithm 37. All these steps will be developed and explained in the following sections. It is also possible to detect internal copies where the copies have been flipped. It only requires to compute descriptors for the flipped patches; this is easily done by computing descriptors on the flipped image and flip the resulting descriptor map. The distance function used for *PatchMath* in this case is the distance from the regular descriptors to the flipped version. Another possible solution is to use flip-invariant descriptors such the one presented in [ZN13] or [XTWZ15] so to avoid two computations of the descriptors; this is not studied in this chapter though.

11.4.1 Median Filter

The first step is to apply a median filter to the displacement map. It smooths the displacement and improves the detection rate by reducing the overall error computed in the following step. The algorithm used for the median filtering is shown in Algorithm 31.

Algorithm 31: medianFilter algorithm

input : A displacement map D, the radius of the filter ρ_m
output : A filtered displacement map $\tilde{D}$

```
1 foreach pixel p in D do
2 |  D̃(p) ← median({D(p') | p' ∈ 𝔻(p, ρ_m)})
```

11.4.2 Error Filter and Detection

A copy-move forgery (as its name indicates) copies a region of the image, possibly rotates it, and pastes it at a different image position. Both regions (the original one and the forged one) can then be matched by an affine transform. The idea presented in [CPV15] is then to compare the field of matches created using *PatchMatch* to the one computed using the best affine transform based on the field of *PatchMatch* matches.

Let $P \in \mathcal{M}^{3\times n}(\mathbb{R})$ be the homogeneous coordinates of a set of points. We set

$$P = \begin{pmatrix} x_1 & \dots & x_n \\ y_1 & \dots & y_n \\ 1 & \dots & 1 \end{pmatrix}. \tag{11.22}$$

Let Δ be the displacement of these same points (still in homogeneous coordinates). Let $A \in \mathcal{M}^{3\times 3}(\mathbb{R})$ be a matrix representing an affine transform. We are looking for a region such that the

transformation error defined by

$$\epsilon(P) = \min_{A \in \mathcal{M}^{3\times3}(\mathbb{R})} \|\Delta - (AP - P)\|^2, \tag{11.23}$$

is small. The best affine transform minimizing Equation (11.23) can be estimated using a simple Least Square Regression, i.e.

$$A = \underset{A' \in \mathcal{M}^{3\times3}(\mathbb{R})}{\arg\min} \|\Delta - (A'P - P)\|^2. \tag{11.24}$$

The solution of Equation (11.24) can be expressed as

$$A = (P + \Delta)P^T(PP^T)^{-1}, \tag{11.25}$$

therefore the error can be estimated without a minimization step by replacing A by its solution from Equation (11.25)

$$\begin{aligned}
\epsilon(P) &= \|\Delta - (AP - P)\|^2 \\
&= \|\Delta - ((P + \Delta)P^T(PP^T)^{-1}P - P)\|^2 \\
&= \|\Delta - \Delta P^T(PP^T)^{-1}P\|^2 \\
&= \|\Delta(I - H)\|^2, \text{ where } H = P^T(PP^T)^{-1}P.
\end{aligned} \tag{11.26}$$

The set of points P used to estimate this error will be a disk of radius ρ_e and the error will be associated to the center of the disk, which delivers an actual error map. In practice the result is independent of the coordinates in P, as long as Δ is set conveniently. Therefore H can be precomputed for all regions. The error detection map is given by thresholding by τ the error map estimated with Equation (11.26). The features and the error filter aren't properly defined on the boundary (it would require to define the behavior of these objects outside the image as well), therefore detections on the boundary are not well defined either. This is why the boundary of the image is removed during the detection step so as to avoid any false detection due to a false matching. The algorithm used to compute the error detection map is summarized in Algorithm 32.

Algorithm 32: errorDetectionFilter algorithm

input : A displacement map D, the radius of the filter ρ_e, τ a threshold, sp the half-size of the patch used to compute the features

output : An error detection map E

$P \leftarrow \mathbb{D}(0, \rho_e)$ `// Define the homogeneous coordinates`

$H \leftarrow P^T(PP^T)^{-1}P$ `// See Equation` (11.26)

$H' \leftarrow I - H$

foreach *pixel* p *in* D **do**

  Let Δ be the displacements corresponding to $\mathbb{D}(p, \rho_e)$

  $e \leftarrow \|\Delta H'\|^2$ `// See Equation` (11.26)

  $E(p) \leftarrow \begin{cases} 1 & \text{if } e < \tau \\ 0 & \text{otherwise} \end{cases}$

Remove any detection at the boundary (of size sp) of E

The next steps of the detection correspond to multiple post-processings trying to minimize false alarms and to adjust the detection mask to the actual forged regions.

11.4.3 Size Filter

The first post-processing step tries to remove "unlucky" matches. Sometimes good matches can be created purely by chance, but in practice these fortuitous regions are very small. The application of a filter which cuts connected components whose area is below a certain threshold removes these "unluckily" matched regions. Such a filter is presented in Algorithm 33.

Algorithm 33: sizeFilter algorithm

input : A detection mask M, area threshold τ_s
output : A filtered detection mask $\tilde{M}$

```
foreach 4-connected component c in M do
    if c is smaller than τ_s pixels then
        M̃(c) ← 0 // Assignment for each pixels in c
```

11.4.4 Minimum Displacement Filter

As a last step to remove false alarms, candidates that are too close in the image plane are filtered out. If the copy is too close to the original, humans detect easily the forgery. Thus, if a detected forged region is too close to its copy, both are considered false alarms and therefore discarded. This idea is synthesized in Algorithm 34. While this post-processing is necessary for the method to work well, it's actually redundant with the exclusion region from *PatchMatch*. Indeed setting τ_m to the maximum between τ_m and τ_d forces all matches of *PatchMatch* to be at a distance superior or equal to τ_d. Therefore there would be no matches to be rejected with this displacement filter. The deliberate choice to still present the displacement filter was made to describe the originally proposed post-processing steps, without mixing ideas from different sections.

Algorithm 34: minDispFilter algorithm

input : A mask of detections M, a displacement map D, τ_d the minimum match distance
output : A filtered mask $\tilde{M}$

```
foreach pixel p in M do
    if the distance in the image between p and D(p) is smaller than τ_d then
        M̃(p) = 0
```

11.4.5 Symmetrization of Detections

Because the process is not entirely symmetric, it can happen that both regions detected as forged have different shape and size. It may even happen that only one of the forged regions is detected while the other one is missing from the detection mask. To avoid this problem, the detection mask is symmetrized using the displacement map. This process is presented in Algorithm 35. This section is necessary even though *PatchMatch* already has a mechanism favoring the symmetry. Indeed *PatchMatch* only incites it in the sense that if the symmetric match isn't a better candidate then it's rejected. Therefore at the end of the *PatchMatch* step, the displacement can be asymmetric (even though in practice it is already quite symmetric). Because a forgery is necessarily symmetric, the final detection map must be symmetric. Therefore the detection regions in the detection map are symmetrized using the method presented in this section.

Algorithm 35: symmetrizationFilter algorithm

input : A detection mask M, a displacement map D
output : A filtered displacement map $\tilde{M}$

foreach *pixel* p *in* M **do**
  $\tilde{M}(D(p)) \leftarrow \max\{M(p), M(D(p))\}$

11.4.6 Dilation

Finally, for the last step, each detection is slightly dilated. Indeed both the median filter and the error filter tend to reduce the size of a detected region. This dilation step mitigates these effects. This step is presented in Algorithm 36.

Algorithm 36: dilationFilter algorithm

input : A mask of detections M, the radius of the filter ρ
output : A dilated mask $\tilde{M}$

foreach *pixel* p *in* M **do**
  $\tilde{M}(p) \leftarrow \max\{M(p') \mid p' \in \mathbb{D}(p, \rho)\}$ // $\mathbb{D}(p, \rho)$ is the set of elements at distance smaller or equal than ρ from p

The complete algorithm regrouping all the steps presented in the previous sections 11.2, 11.3 and 11.4 is summarized in Algorithm 37.

Algorithm 37: Copy-move forgery detection

input : A suspect I, sp half-size of the patches, n_i number of *PatchMatch* iteration, τ_m minimum match distance for *PatchMatch*, the radius ρ_m, respectively ρ_e, for the median, respectively error computation, τ error threshold, τ_s size threshold, distance threshold τ_d
output : A final detection mask F and binary forgery decision

$Z \leftarrow ZernikeMoment(I, sp)$ // See Algorithm 19
$D \leftarrow Patchmatch(Z, n_i)$ // See Algorithm 31
$\tilde{D} \leftarrow medianFilter(D, \rho_m)$ // See Algorithm 31
$E \leftarrow errorDetectionFilter(\tilde{D}, \rho_e, \tau)$ // See Algorithm 32
$\tilde{M}_1 \leftarrow sizeFilter(E, \tau_s)$ // See Algorithm 33
$\tilde{M}_2 \leftarrow minDispFilter(\tilde{M}_1, \tau_d)$ // See Algorithm 34
$\tilde{M}_3 \leftarrow symmetrizationFilter(\tilde{M}_2)$ // See Algorithm 35
$F \leftarrow dilationFilter(\tilde{M}_2, \rho_m + \rho_e)$ // See Algorithm 36
return F *and whether* F *is empty or not*

11.5 Experiments

In this section, results showing both the strengths and the limits of the method are presented. The displacement results presented in this section will have the same visualization. The color scheme used to represent the direction and magnitude of the displacement is shown in Figure 11.4.

Maximum order of the Zernike descriptors o	5
Half-size of patch used to compute the moments sp	8
Number of *PatchMatch* iterations N	8
Minimum distance between two *PatchMatch* matches τ_m	8
Minimum distance between two clones τ_d	50
Error threshold τ	300
Minimum size of a clone τ_s	1200
Radius of the median filter ρ_m	4
Radius of the error filter ρ_e	6

Table 11.1: Value of the parameters used during the computation of the results in Section 11.5.1.

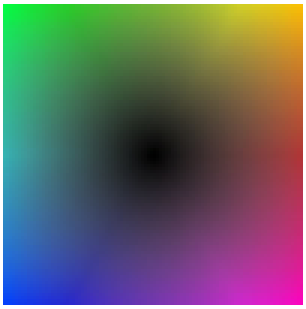


Figure 11.4: Color scheme representing the direction and magnitude of the displacement

11.5.1 Using Zernike Moments

The parameters that will be used for the experiments are those suggested in [CPV15]. They are reminded in Table 11.1. The different images used in this section come from the FAU database presented by Christlein et al. in [CRJ+12]. The first example, shown in Figure 11.5, is a straightforward example illustrating the quality of the detections by the method in the simplest case. We can see that every step performs as expected and provides an accurate estimation of the forged regions. It is also important to test the method's resilience to false positives. This is why we applied the exact same method to an intact image containing several instances of a similar object, shown in Figure 11.5. The result confirms the absence of detection in this case even though some regions could have appeared forged due to their similarity.

One of the requirements of the method was to be rotation resilient. An example of forgery with a rotation is presented in Figure 11.5. It actually confirms that the detection succeeds when a rotation has been applied. Even though only a single example is shown, more tests confirming the rotation resilience have been done with varying degrees of rotation. The same experiments have also been done using basic Cartesian Zernike moments in order to see the importance of the resampled version presented in Section 11.2. In this case, also presented in Figure 11.5, the detection is much worse (only a much smaller region is detected) than when the resampled Zernike moments are used.

We also tested other classic perturbations such as the addition of noise. Figure 11.6 presents a result with a small amount of noise added after the forgery. The method still works in presence of small noise. The addition of a large noise in a forgery would make the image suspicious anyway. Figure 11.6 shows the importance of also taking account flipped copies.

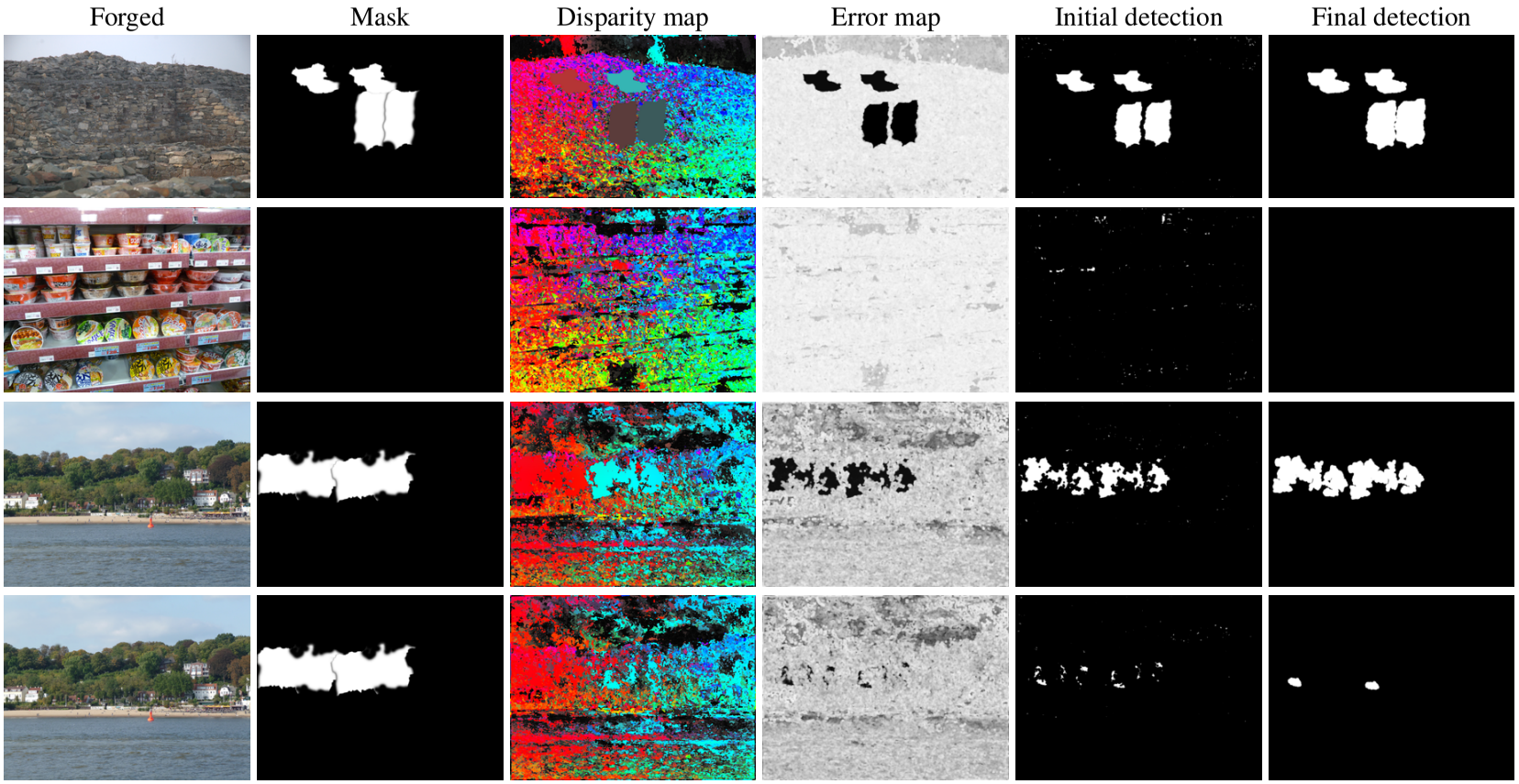


Figure 11.5: Result of the method using Zernike moments on (from top to bottom): a forged image with a forgery applying only a translation, a clean image, a forged image with a forgery applying a translation and a ten degrees rotation (resampled Zernike moments), a forged image with a forgery applying a translation and a ten degrees rotation (basic Cartesian Zernike moments).

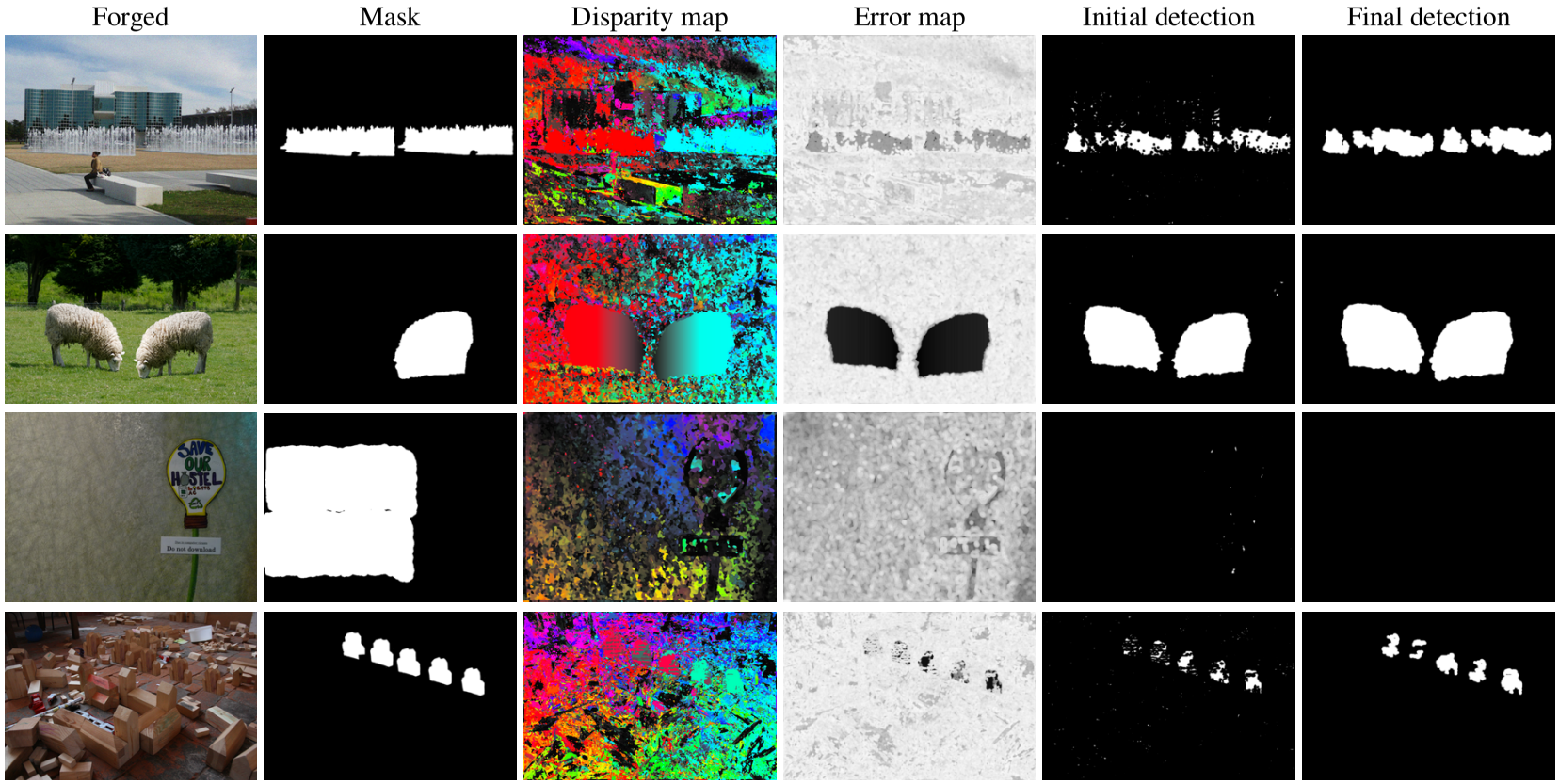


Figure 11.6: Result of the method using Zernike moments on (from top to bottom): a forged image with a forgery applying a translation on top of which a noise of standard deviation 25 was added, a forged image with a flipped copy, a forged image for a forgery applying a Poisson editing, a forged image for a forgery applying translations when a same object is copied at multiple different positions.

The next example of Figure 11.6 is no longer a simple copy-move forgery but an actual editing of the image. Indeed the copied section was integrated using the Poisson editing method presented in [PGB03] and implemented in [DMFML16]. Poisson editing is quite adapted to forgery as it blends nicely the pasted region in the image. This time, the algorithm actually fails. This shows the limitation not of the method itself but of the descriptors, namely the amplitude of Zernike moments. Those descriptors achieve the wanted invariance but at the same time they are extremely rigid. With appropriate descriptors, the exact same methodology could be applied and should yield good detections. Removing the Zernike moment of order 0 from the list of descriptors used (which actually corresponds to the mean of the local region) didn't improve the results for Poisson editing.

Another limit of the method is the presence of multiple copies. In such a case, the *PatchMatch* displacement field is at risk of splitting between the multiple copies, which increases the error and decreases the chance of an actual detection. Figure 11.6 shows an example where this problem arises. In that case, parts of each of the copies are actually detected but none is entirely detected for the reasons explained above.

11.5.2 Using Dense SIFT Descriptors

Although the method performs as expected, as shown in Section 11.5.1, it is disappointing that forgeries such as Poisson editing are not detected. For this reason, the same experiments have been repeated after changing the Zernike moments by dense SIFT descriptors. SIFT descriptors, presented by Lowe in [Low99], have been shown to be state of the art descriptors in image matching thanks to their invariance to noise, changes of lighting or to small changes of viewpoint. In particular, they are rotation invariant, just like the Zernike moments. A dense version of these descriptors can be computed by considering all patches instead of those marked by points of interest. The dense version was the one used for the experiments. The VLFeat implementation of dense SIFT from [VF08] was used. The parameters are listed in Table 11.2. Figures 11.7 and 11.8 show the results of the same experiments shown in Section 11.5.1. As one can see, dense SIFT descriptors perform as well as the Zernike moments on the successful experiments but also successfully detect Poisson editing. Nevertheless, dense SIFT descriptors seem to incur into a higher risk of false positives, see Figure 11.7 (clean image) and 11.8 (Poisson).

Number of SIFT bins	4
Size of SIFT bins	5
Number of *PatchMatch* iterations N	8
Minimum distance between two *PatchMatch* matches τ_m	8
Minimum distance between two clones τ_d	50
Error threshold τ	300
Minimum size of a clone τ_s	1200
Radius of the median filter ρ_m	4
Radius of the error filter ρ_e	6

Table 11.2: Value of the parameters used during the computation of the results in Section 11.5.2.

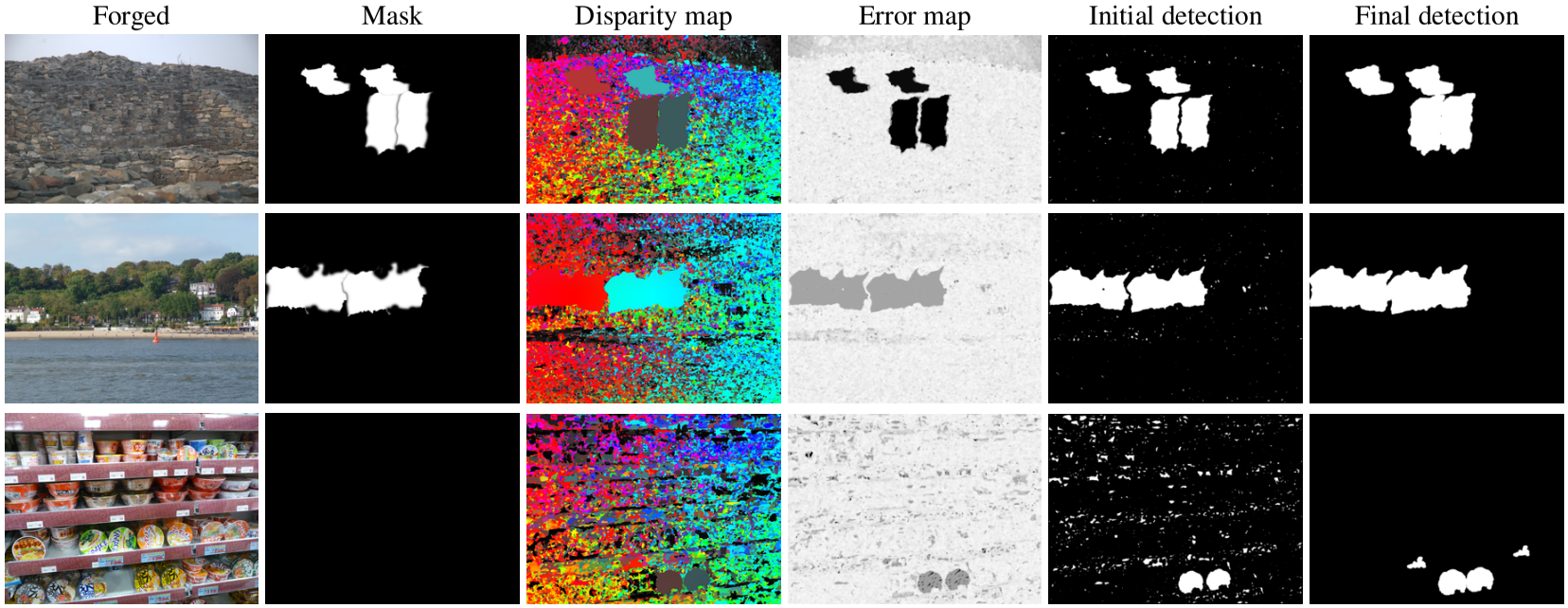


Figure 11.7: Result of the method using dense SIFT on (from top to bottom): a forged image with a forgery applying only a translation, a forged image with a forgery applying a translation and a ten degrees rotation, a clean image.

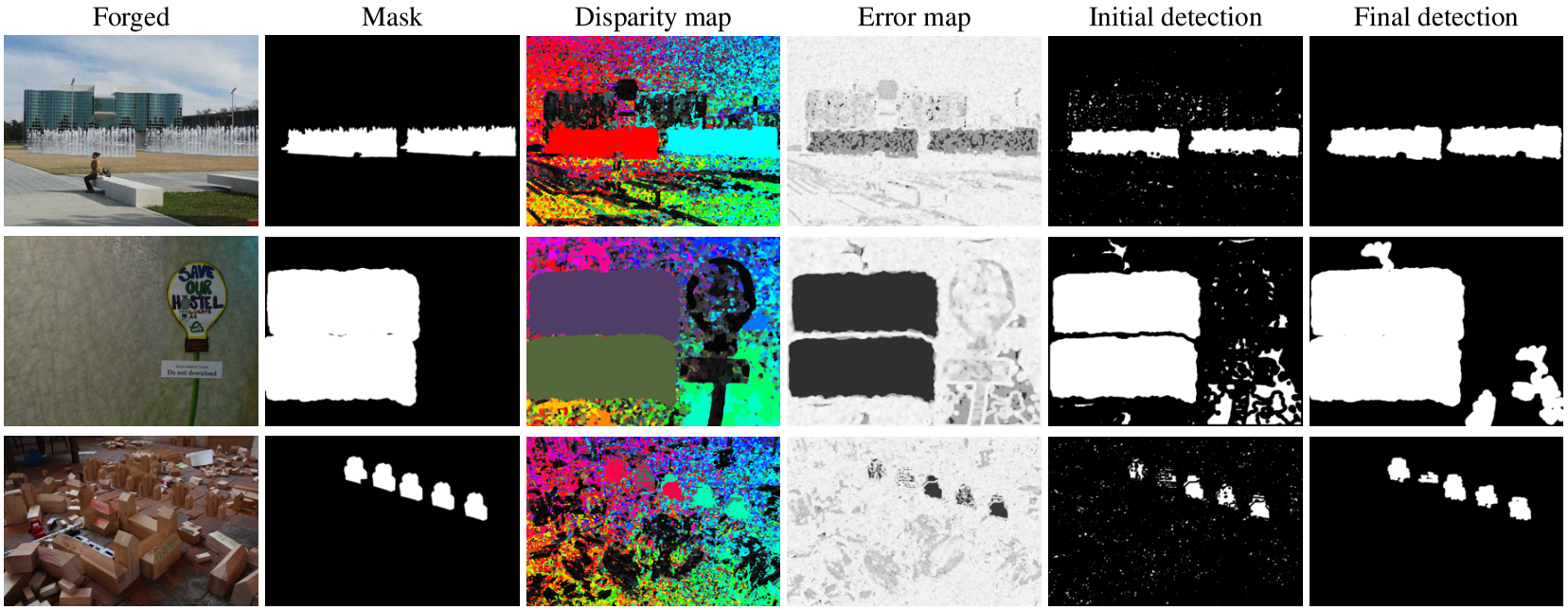


Figure 11.8: Result of the method using dense SIFT on (from top to bottom): a forged image with a forgery applying a translation on top of which a noise of standard deviation 25 was added, a forged image for a forgery applying a Poisson editing, a forged image for a forgery applying translations when a same object is copied at multiple different positions.

11.6 Conclusion

The forgery detection method analyzed in the present chapter proves to be an efficient detector. One of its major advantages is its resilience to rotations. It is also robust to image perturbations such as the addition of noise. Nevertheless, it is not deprived of drawbacks. In particular, the results are particularly disappointing when considering multiple copies of an object or when the forgery has been performed by a classic copy-paste method such as Poisson editing. A better choice for the descriptor should be able to fix this deficiency of the method though. Dense SIFT descriptors were considered as an alternative to improve the detection of forgeries performed by Poisson editing. Even though these descriptors allow for better detections, they are more likely to produce false positives. Nevertheless, this shows that the method can be modified and improved by varying the descriptors. The only drawbacks found – which are mainly linked to the usage of *PatchMatch* – is the inability of the method to detect multiple copies but also to not distinguish well similar objects from actual copies.

12 Detection of copy-paste forgeries based on sparse descriptors

Detecting reliably copy-move forgeries is difficult because images do contain similar objects. The question is : how to discard natural image self-similarities while still detecting copy-moved parts as being “unnaturally similar”? Copy-move may have been performed after a rotation, a change of scale and followed by JPEG compression or the addition of noise. For this reason, we base our method on SIFT, which provides sparse keypoints with scale, rotation and illumination invariant descriptors. To discriminate natural descriptor matches from artificial ones, we introduce an *a contrario* method which gives theoretical guarantees on the number of false alarms. We validate our method on several databases. Being fully unsupervised it can be integrated into any generic automated image tampering detection pipeline.

12.1 Introduction

Photo and video editing includes the insertion or removal of parts of the image, often performed by internal or external copy-move operations. The Poisson editing technique [PGB03, DMFML16] allows for seamless insertions and is now routinely used for special effects in movies, in software like Photoshop or in popular mobile phone applications. Most editing operations are driven by aesthetic goals. Yet their usage can easily become malicious and help forging false evidence, fake news, or alter results in scientific publications [BCF16].

It is therefore of primary importance to provide public and professionals with reliable scientific tools detecting traces of any intentional alteration of a photograph. Several different techniques are relevant here: image splicing (internal or external) can be detected through its local alterations of the compression encoding and of the JPEG blocks [LHTT09,CGFY12,NGvGCM18], its inconsistent demosaicking traces [PF05, FBDRP12, BGvGM18], or directly [NCS04, HC07, HLOE18]. Methods tracking other features such as noise inconsistencies, lightning inconsistencies, chromatic aberration inconsistencies, etc. were listed in the broad review [Far09].

This chapter focuses on a specific type of image splicing called “copy-move". As its name indicates it consists in copying a region of the image and pasting it somewhere else. Rotation, scaling, change of contrasts and other manipulations are sometimes applied to the piece being copied before pasting it. The method can be used to replicate objects, but sometimes also to hide an object by a texture borrowed elsewhere in the image. Copy-move detection methods can be divided into two main categories: Block-based and keypoint-based. The block-based approaches try to match regions by blocks. In order to match the blocks more easily and more efficiently it is frequent to represent the block in a compact form by dimensionality reduction, e.g. with PCA,

DCT [CGFY12] or DWT [LWTS07]. The compact representation may also ensure the invariance of the detection to rotations by using Zernike moments [CPV15, Ehr18] or a similarity invariance with the Fourier-Mellin transform [BSM09, LY10]. These methods generally manage to detect the forged regions, but are computationally demanding. Instead of directly trying to match blocks, featured-based methods compute sets of keypoints and then match these keypoints. Many of these methods are based on SIFT [ABC+10, ABM10] or SURF [BJGY10]. The descriptors associated to the keypoints are invariant to rotation, scaling and even moderate affine distortions. Yet, precisely out of too much robustness, these methods may cause false detections when similar objects are present in an image. As argued in [WZS+16], most methods therefore suffer from a false positive problem caused by the occurrence of "natural" self-similarity. This is the problem that we attempt to tackle here.

Section 12.2 introduces our method and Section 12.3 shows experimental results on different datasets. Finally perspectives are going to be presented in Section 12.4.

12.2 Copy-move matching with SIFT-like matching

Like in the SIFT algorithm [Low99] we start by computing a set of sparse keypoints. These keypoints are usually located in textured regions. Then a descriptor is associated to each of these keypoints. Finally the descriptors are matched to each other to define the detection. These three steps are summarized in the next paragraphs.

12.2.1 Keypoints

The keypoints correspond to the extrema of the normalized Laplacian scale-space. In practice they are computed using differences of Gaussians. The positions of the maxima are then found for each scale. To each keypoint is associated a scale and a principal orientation. A more detailed analysis can be found in [ROD14].

12.2.2 Descriptors

This first, classic, SIFT step gives a list $\mathcal{K} = (k_i)$ of keypoints. From each of these keypoints (consisting of a spatial position, a scale and an orientation), a square patch p_i of size $(N+2) \times (N+2)$ can be sampled. The gradients in both directions are then computed from these patches yielding an $N \times N$ gradient patch $\mathcal{D}^i$ with vector values $(\frac{\partial p_i}{\partial x}, \frac{\partial p_i}{\partial y})$.

Contrary to SIFT, we keep these matrices for the matching step. Indeed, using histograms of gradients (HOGs) to represent the gradient patch would be too robust a representation and lead to the detection of natural repetitions. Hence, following [GP15] and [RGvG18]. we encode the key point k_i by its gradient patch $\mathcal{D}_i$. This allows for an invariance to uniform illumination changes. In SIFT, the descriptors are computed on a grayscale version of the image for matching applications. In the forgery case, it is interesting to consider all information available. So our gradient descriptors keep three channels, one for each color. For simplicity our matching step will be presented using a grayscale descriptor, but extends immediately to color as well. Color descriptors will be used for the experiments in Section 12.3.

12.2.3 Matching

Two naturally similar objects are rarely exactly similar. This is because there are always differences in their illumination in a real scene, and physical differences that do not necessarily catch the eye, in addition to the acquisition noise. Our matching process takes advantage of these serious variations to discriminate between similar objects and digital copies. Consider two keypoints k_i and k_j located respectively on the original object and on the forged copy, so

Figure 12.1: On the left image the two objects are similar but not digital copies of each other. On the right, one is a digital copy of the other. The patches shown below each respective image correspond to the red dots in the images. They show that a difference is visible at this level and therefore the descriptors can be discriminated. This is why the detection method can discard genuinely similar objects.

that they match with a regular matching method such as SIFT [Low99] or [RGvG18]. In that case the descriptors $\mathcal{D}^i$ and $\mathcal{D}^j$ associated to these keypoints should be *exactly* the same, namely $\forall k,l \in \{1,\dots,N\}, \mathcal{D}^i_{k,l} = \mathcal{D}^j_{k,l}$. Of course this perfect quality is not reached in practice. Several copy-move steps could introduce small differences such as: the interpolation due to a rotation or zoom or even a post-processing step such as the addition of noise and/or a compression after forgery. Nevertheless, we can enforce a very close match between each part of the descriptors by an exigence like $\forall k,l \in \{1,\dots,N\}, \|\mathcal{D}^i_{k,l} - \mathcal{D}^j_{k,l}\|_2^2 \leqslant \tau$. For the distance d_{max} defined by

$$d_{max}(\mathcal{D}^i, \mathcal{D}^j) = \max_{k,l \in \{1,\dots,N\}} \|\mathcal{D}^i_{k,l} - \mathcal{D}^j_{k,l}\|_2^2, \tag{12.1}$$

the suspicious match test is simply $d_{max}(\mathcal{D}^i, \mathcal{D}^j) \leqslant \tau$. This corresponds to a stricter version of matching than the one presented to match patches in Chapter 3.

The key question is to fix the right detection threshold τ, to have a matching criterion that rejects genuinely similar objects while still detecting well copy-move forgeries. This threshold can be computed rigorously using the *a-contrario* theory [DMM07] which is a probabilistic formalization of the *non-accidentalness* principle [Low85]. This principle has shown its practical use for detection purposes such as segment detection [VGJMR10], vanishing points detection [LGvGRM14] and anomaly detection [DEMD18]. The *a-contrario* theory provides a way to compute automatically detection thresholds while having a control on the number of false alarms (NFA). It replaces the usual p-value by drawing into account the number of tests and therefore controlling the overall number of false alarms in a given detection task. The method only requires a simple

a contrario stochastic model on the perturbation. We will consider for now that $\mathcal{D}^i$ and $\mathcal{D}^j$ are derived from the same patch but one of them has been corrupted by Gaussian noise of variance σ^2 *i.e.* since the descriptors consist of gradients $\forall k, l, \mathcal{D}^j_{k,l} = \mathcal{D}^i_{k,l} + n_{k,l}$ where $n_{k,l} \sim \mathcal{N}(0, 2\sigma^2)$ and are independent. Matching both descriptors requires that $\max_{k,l} n^2_{k,l} \leqslant \tau$. The probability of matching in this case is then

$$\mathbb{P}\left(\max_{k,l} n^2_{k,l} \leqslant \tau\right) = \prod_{k,l} \mathbb{P}\left(n^2_{k,l} \leqslant \tau\right) = \mathbb{P}\left(n \leqslant \frac{\tau}{2\sigma^2}\right)^{N^2} \tag{12.2}$$

where n follows a χ^2 distribution with 1 degree of freedom. We can therefore control the number of false detections by choosing the proper τ according to

$$\tau = 2\sigma^2 \times chi2inv\left(\sqrt[N^2]{\frac{\epsilon}{N_{tests}}}\right), \tag{12.3}$$

where ϵ is the number of false alarms per N_{tests} number of tests and $chi2inv$ is inverse of the χ^2 cumulative distribution function. The main point of formula (12.3) is that it reduces the initial method dependency on many detection parameters to just one, namely σ. We can argue that this last one is not critical. Indeed, even though the dependency on σ is strong, as long as the degradation is not too large, there will be a scale in which σ is small enough so the detection will work: indeed σ is divided by two at each octave in the SIFT method. For example, this exigent threshold can work for a noise of 4, but requires the tampered area to be four times larger for a detection. It might be objected that zooming down also makes naturally similar objects become more similar. Yet our experiments indicate that this is not the case, indeed their small but significant differences encompass all scales. To summarize, granting that we allow for one false detection on average on a set of images, the method is parameterless as it adapts to the number of tests and to the patch size. Of course it might be coupled with an automatic noise estimator to give an good guess of σ. Assuming a perturbation noise of variance $\sigma^2 = 1$ and the use of descriptors of size $4 \times 4 \times 3$ (derived from 6×6 color patches), a number of false alarms of $\epsilon = 1$ and testing on 100 images with on average 50 keypoints (this corresponds to the COVERAGE dataset presented in Section 12.3), Equation (12.3) gives $\tau = 2.9$. The advantage of using color descriptors in this case is either to increase the size of the descriptor (allowing for a larger τ and detecting more) or to reduce the spatial size for a same size of descriptor (allowing to detect smaller forgeries).

An interesting side effect is that this test is really fast to compute. Indeed to detect forgeries each keypoint must to compared against all others. Since we are comparing keypoints inside a single image this gives $\frac{(K-1)(K-2)}{2}$ pairs to be tested, where K is the number of descriptors. (Of course all descriptor self-matches are discarded). For large images the computation of the distance becomes quickly a bottleneck for distances that are costly. In our case it is not necessary to compute d_{max} before doing the test, the test can be done during the computation of d_{max} which allows for early stopping. Since most keypoints won't match, the number of operations done per comparison is in practice much smaller than the size N^2 of the descriptor. An experimental verification of this fact is made in Section 12.3.

Finally we need to take into account all possible flips for the forged regions. While the matching process doesn't detect flips it is possible to still detect them at the cost of a few more computations. The modified distance to test flips is then

$$d_{flip}(\mathcal{D}^i, \mathcal{D}^j) = \max_{k,l} \left(x^i_{k,l} - x^j_{k,l}\right)^2 + \left(y^i_{k,l} + y^j_{N-k+1,l}\right)^2 \tag{12.4}$$

where $D_{k,l} = (x_{k,l}, y_{k,l})$. Indeed when flipped, the indexes in one direction are reversed but also the gradients in that direction are opposite. Thanks to the rotation invariance all flips are taken

Dataset	Method	True detections	False detections
COVERAGE	Proposed	43%	1%
	Previous	50.5%	-
GRIP	Proposed	70%	1.3%
	Previous	71%	-
IM	Proposed	81.2%	0%
	Previous	75%	-
IM JPEG80	Proposed	64.6%	0%
IM NOISE20	Proposed	79.2%	0%

Table 12.1: Detection statistics on the different datasets compared to the reported results from [WZS+16]. The proposed method achieves similar true positive detections for a very limited number of false detections. The only false detection is shown in 12.3. The false detections are computed on the original images (with no forgeries).

into account by just testing the flip in one direction (in our case in the y direction). In the end, we test each pair of keypoints with both distances to take into account flips. Having to do twice the computation is not a problem in practice as each test is very efficient.

12.3 Experiments

In this section the images are all shown in grayscale even though they are originally in color. This allows for a better visualization of the matches. Nevertheless, the descriptors were color descriptors. We present results on three different datasets: GRIP [CPV15], Image Manipulation (IM) [CRJ+12] and COVERAGE [WZS+16] which is the dataset that inspired this study as it focuses on distinguishing forgeries from similar but genuine objects. All images shown in this section come from these datasets.

We decided to use descriptors of size $3 \times 8 \times 8$ for the IM dataset and $3 \times 4 \times 4$ otherwise as the images from the IM dataset are much larger than the ones from the other datasets. Indeed the size of the descriptors needs be chosen so to be smaller than the expected size of the forged regions. Each time the threshold was computed using Equation (12.3) from Section 12.2.3. We also verified that the number of comparisons done to compare two descriptors was much smaller than the size of a descriptor. For example for the image shown in Figure 12.1, of size 424×421 and containing 207 descriptors only 1.1 comparisons were necessary on average for a descriptor size of $3 \times 6 \times 6 = 108$ that is almost the size of the descriptor used for SIFT. Thus the detection is really fast even for large images with a large number of keypoints.

Table 12.1 shows that while the methods focuses on being robust to similar objects and reduces as much as possible false detections, it is actually competitive with previous keypoint based methods. Moreover, the number of false alarm is definitely under control : only very little false alarms were found in all three datasets. One of these false detection is shown in Figure 12.3. We also verified that while robust to similar objects (and therefore very precise) the method still is robust to reasonable noise and compression.

Figure 12.2 shows different examples of successful detections. The method is able to detect well rotation, uniform illumination changes, scaling and compression. As can be seen, the more

Figure 12.2: Examples of forgeries that were successfully detected in the different datasets. From the top to bottom, left to right: an example with a rotation, a change of illumination, a change of scale and with JPEG compression.

texture the forged region has the easier it is to detect. This is because a textured region will generate more keypoints and therefore will increase its chances of matching.

Figure 12.3 shows several failure examples. Most failures come from the fact that the method can't deal with more severe distortions such as a tilt or non-uniform illumination change. The method also fails to detect flat regions, as no keypoints are computed on these regions. As for the false detection, it does not contradict the a contrario model. We requested at most $\epsilon = 1$ false detection per 100 images with the threshold given in Section 12.2.3, and we found one with 200 images tested for the COVERAGE dataset.

12.4 Perspectives

In this chapter we have presented an unsupervised method to detect copy-move forgeries that is not only invariant to rotation, scaling and global change of illumination, but also robust to the

Figure 12.3: Examples of forgeries that were not successfully detect in the different datasets. From top to bottom, left to right: an example where the forged region is completely flat, with a non-uniform change of illumination, with a tilt applied and the false detection.

presence of similar but genuinely different objects or regions. The method, being parameterless and very fast, can be included in the necessary long series of tampering tests applied to a suspicious image.

The limits of the method are closely linked to its strength. Because it is robust to the presence of naturally similar objects, it is less reliable in case of large degradation of a copied digital ones. We nevertheless found that the method is robust enough to usual noise and compression levels. An image that has been degraded too much is suspicious anyway, since nowadays the quality of an image taken with a mobile is very good. Thus only images with a good enough quality should be tested. Highly degraded images would be anyway suspicious regardless of any such more sophisticated examination. The second limit is the usage of sparse keypoints. These keypoints are only computed in regions that are contrasted enough (non-flat areas) which means that forgeries in these regions might not detected. Finally matching keypoints give anchor points and do not delimit forged regions precisely. A natural extension of the method would be to extract the forged regions from the anchor points while still keeping a good control over the number of false detections. Finally coupling the method with a noise estimator could arguably make it still more discriminant.

Conclusion

The main focus of this dissertation was video denoising, with detours to other problems that can be approached by applying some of the same techniques used for video denoising. We first studied patch-based methods, which were the state-of-the-art at the beginning of the thesis. We looked at many different aspects of patch-based methods: the models for similar patches, patch shapes and the patch search. We took an in-depth look at a popular video denoising method called VBM3D in Chapter 1. We also looked at how to integrate new improvements such as multi-scale, optical flow and spatio-temporal patches. In particular, VBM3D combined with spatio-temporal patches and a patch-search guided with an optical flow gives very competitive results even when compared to the latest state of the art, especially for higher noises. VBM3D and its variants provide an interesting trade-off between denoising quality and computational complexity.

We then turned our attention to the patch search, a rather overlooked aspect of patch-based methods for video. We studied the performance gain obtained by expanding the local patch search into a global one for patch-based video denoising algorithms in Chapter 2. With the global search, the patches found can follow long trajectories in the video, thus fully benefiting from the temporal redundancy of videos. Thanks to an efficient approximate search, we proposed global video denoising methods by extending to video BM3D and NL-Bayes, two patch-based image denoising algorithms. We obtained a significant boost on the denoising performance. This performance boost is only slightly more costly than a local exhaustive search, including the time spent building the tree, thanks to a simple parallelization.

Finally, we focused on how to obtain real-time performance with patch-based methods: First by proposing novel recursive algorithms, then GPU implementations of existing methods. A global patch search such as the one presented in Chapter 2 is still too costly for practical video denoising applications. This is why we presented a novel patch-based video denoising in Chapter 3 combining temporal filtering and non-locality. The temporal filtering is carried out by Kalman filters whose parameters are estimated non-locally. The resulting method is recursive: to denoise one frame, it uses solely information from the previous frame. Results show a performance comparable to more complex patch-based methods that use around ten frames to denoise each single frame. The main limitation of the method is that it relies heavily on the optical flow. Slight errors in registration cause a drop in PSNR. Nevertheless, visual quality of the result is superior, both in terms of details and temporal consistency, to algorithms with comparable running times.

In Chapter 4, we proposed a GPU implementation of NL-means, BM3D and VBM3D to further reduce computation times. This chapter focuses on the technical challenges of such implementations and how to overcome them. We also proposed compromises for the default parameters to get an even larger speedup with only a minor penalty for the denoising performance. This work paves the way to new uses of these algorithms in time constrained environments and for practical applications.

During the course of this thesis, the image processing community experienced a revolution as it was demonstrated that convolutional neural networks, trained with supervision, achieved much

better performance than traditional methods across a wide range of image restoration tasks, including image denoising. This lead us to shift the focus of our research on how to apply those machine learning methods to video denoising. In particular, we described an effective way of incorporating temporal non-local information into a CNN for video denoising in Chapter 5. The proposed method computes for each image patch the n most similar neighbors on a spatio-temporal window and gathers the value of the central pixel of each similar patch to form a non-local feature vector which is given to a CNN. Our method yields a significant gain compared to other CNN approaches. It even has similar performance to the best non-CNN method evaluated, even outperforming them on the largest of our test datasets.

With the framework presented in Chapter 6, we show that a single video is often enough to "train" a denoising network, removing the need for a dataset of images. By applying a simple frame-to-frame training on a generic pre-trained network (for example a DnCNN network trained for additive Gaussian noise with fixed standard deviation), we successfully denoised a wide range of different noise models even though the network has *never* seen the video nor the noise model before its fine-tuning. This opens the possibility to easily process data from any unknown origin. We think that the current fine-tuning process can still be improved. First, given that the application is video denoising, it is expected that better results will be achieved by a video denoising network (the DnCNN network processes each frame independent of the others). Using the temporal information could improve the denoising quality, just like video denoising methods improve over frame-by-frame image denoising methods, but also might stabilize the variance of the result for the on-line fine-tuning. Recent works have also shown the possibility of real time fine-tuning [TRJ+19, TTP+19] paving the way to improve computation time.

Fine-tuning on a single video is very useful for data that has been processed by an unknown pipeline but it can also be used on raw data, *i.e.* mosaicked raw video. Indeed, it is possible to both demosaick and denoise at the same time with a single network. While videos are rarely shot in raw, this approach is particularly useful for image bursts that are more and more popular, especially with mobile phones. We have seen that the CNN based demosaicking methods beat by almost two decibels the best human-crafted methods while being faster by one order of magnitude in Chapter 7. To reach this performance, they did not rely on the clever human techniques established by the anterior state of the art, but simply applied rather standard CNN architectures. This success is explainable. The first reason is that human-crafted algorithms rely on the iteration of nonlinear local filters, the most complex one, ARI, having more than 20 such iterations. But deep convolution networks have the same structure and are in addition scalable in depth and number of filters, until they reach the best performance. Furthermore, human-crafted methods have been designed to avoid certain artifacts, probably to the cost of losing in PSNR. CNNs, on the contrary, can be taught to learn intricate casuistry by the variety of presented images, and to keep a memory of complex image statistics. Being trained to reach the best PSNR, they definitely reach that goal. Finally, the Gharbi et al. method succeeded in biasing the learning process so as to avoid completely the most annoying moiré and zipper effects, thus also achieving the most acute requirement of demosaicking methods, while still retaining the best PNSR. This performance arguably seals the destiny of human-crafted methods on this subject.

In Chapter 8, we proposed a novel way of training demosaicking neural networks without any RGB ground truth. We use instead other mosaicked data of the same scene (such as from a burst of images). Based on it and on recent neural network advances, we proposed a method to train jointly demosaicking and denoising with bursts of noisy raw images. We showed that fine-tuning on a given burst boosts the reconstruction performance. Clipped noise, a hard problem, is handled natively. It also presents a specific case where overfitting a network to the training data is valuable. Since we do not expect generalization there's only benefits from this overfitting. While the process of fine-tuning is costly and might not always be useful, training without ground truth

is always an improvement to regular training. We hope our work can lead to new camera pipeline calibration procedures, and general improvement of the image quality when a burst is available.

The last part tackled a different set of problems, namely detection. In particular, we looked at anomaly detection in images and forgery detection and how these problems can be linked to denoising. We have first shown in Chapter 9 that anomalies are easier detected on a residual image, computed by removing the self-similar component, and then performing hypothesis testing. It is reassuring to see that our method finds all anomalies proposed in the literature with very low NFA. In addition, we have experimentally shown that the method verifies the non-accidentalness principle: no anomalies are detected in white noise. We plan to extend the method to videos, by analyzing anomalies in the motion field.

We then presented a theoretical analysis of the convergence of PatchMatch algorithms in Chapter 10. For an energy level ε, we show that the probability of having an energy above ε converges to zero with the number of iterations, and we provide worst case bounds on the convergence rate. Our analysis applies to the case of k nearest neighbors, and to most variants of PatchMatch proposed in the literature. We give specific bounds for two of these algorithms: the original PatchMatch [BSFG09a, BSGF10] and CHS [KA11]. For the case of the original PatchMatch (with $k = 1$) we validate our results by comparing the predicted convergence rate with the one found in practice. The setting of our framework is rather general: the task of patch matching is viewed as an optimization problem where the goal is to minimize several non-convex energies U_x over the same domain. This might allow the application of these techniques to similar optimizations problems in other areas. For the case of matching patches, it would interesting to precise the link between the regularity of the images and the convergence rate, at least for certain simple models of images.

The forgery detection method analyzed in Chapter 11 takes advantage of *PatchMatch*. One of its major improvements is its resilience to rotations. It is also robust to image perturbations such as the addition of noise. Nevertheless, it is not deprived of drawbacks. In particular, the results are particularly disappointing when considering multiple copies of an object or when the forgery has been performed by a classic copy-paste method such as Poisson editing. A better choice for the descriptor should be able to fix this deficiency of the method though. Dense SIFT descriptors were considered as an alternative to improve the detection of forgeries performed by Poisson editing. Even though these descriptors allow for better detection, they are more likely to produce false positives. Nevertheless, this shows that the method can be modified and improved by varying the descriptors. The only drawbacks found – which are mainly linked to the usage of *PatchMatch* – is the inability of the method to detect multiple copies but also to not distinguish well similar objects from actual copies.

Finally, in Chapter 12 we have presented an unsupervised method to detect copy-move forgeries that is not only invariant to rotation, scaling and global change of illumination, but also robust to the presence of similar but genuinely different objects or regions. The method, being parameter-less and very fast, can be included in the necessary long series of tampering tests applied to a suspicious image. The limits of the method are closely linked to its strength. Because it is robust to the presence of naturally similar objects, it is less reliable in case of large degradations. We nevertheless found that the method is robust enough to usual noise and compression levels. An image that has been degraded too much is suspicious anyway since nowadays the quality of an image taken with a mobile is very good. Thus only images with a good enough quality should be tested. Highly degraded images would be anyway suspicious regardless of a sophisticated examination. The second limit is the usage of sparse keypoints. These keypoints are only computed in regions that are contrasted enough (non-flat areas) which means that forgeries in these regions might not detected. Finally matching keypoints give anchor points and do not delimit forged regions precisely. A natural extension of the method would be to extract the forged regions from

the anchor points while still keeping a good control over the number of false detections. Finally coupling the method with a noise estimator could arguably make it still more discriminant.

A Analysis of PatchMatch (proofs)

A.1 Generic fast patch matching algorithm

A.1.1 Random search

In the following, we will say that $\eta \in \{U_z \geqslant \alpha\}$, η being a k-set of elements, if and only if $U_z(\eta) \geqslant \alpha$ (U_z being defined in Def. (10.1)).

Lemma 11.1. *For all $z \in \boldsymbol{A}$, $C(z, .)$ is a non-increasing function such that for $\varepsilon > 0$, $C(z, \varepsilon) \in [0, 1[$ and $C(z, \varepsilon) = 1$ for $\varepsilon \leqslant 0$.*

Proof. Let us recall the definition of C:

$$C(z, \alpha) = \sup_{\eta \in \{U_z \geqslant \alpha\}} Q(\eta, \{U_z \geqslant \alpha\}). \tag{A.1}$$

The function $C(z, \cdot)$ is a supremum of probabilities (in fact it is a maximum because the set $\{U_z \geqslant \varepsilon\}$ is finite). We remind the property for a candidate η, $Q(\eta, L) < 1$ if the set of acceptable list of candidates L is not empty. Thus $C(z, \varepsilon) \in [0, 1[$, for all $z \in \mathbf{A}$ and $\varepsilon > 0$. Since $\inf U_z = 0$, the probability of transitioning to a negative energy is 0, and thus $C(z, \varepsilon) = 1$ for $\varepsilon \leqslant 0$. If $\varepsilon_1, \varepsilon_2 \in \mathbb{R}_+$ are such that $\varepsilon_1 \geqslant \varepsilon_2$, we have $\{U_z \geqslant \varepsilon_1\} \subseteq \{U_z \geqslant \varepsilon_2\}$. Therefore for all η a set of candidates of $\mathbf{B}$, because Q is a stochastic kernel, $Q(\eta, \{U_z \geqslant \varepsilon_1\}) \leqslant Q(\eta, \{U_z \geqslant \varepsilon_2\})$. This implies that

$$\sup_{\eta \in \{U_z \geqslant \varepsilon_1\}} Q(\eta, \{U_z \geqslant \varepsilon_1\}) \leqslant \sup_{\eta \in \{U_z \geqslant \varepsilon_1\}} Q(\eta, \{U_z \geqslant \varepsilon_2\}) \leqslant \sup_{\eta \in \{U_z \geqslant \varepsilon_2\}} Q(\eta, \{U_z \geqslant \varepsilon_2\}). \tag{A.2}$$

and therefore $C(z, \varepsilon_1) \leqslant C(z, \varepsilon_2)$. □

A.2 Convergence of the patch matching algorithms

For the proof of Lemma 10.2, we will use the following lemma and its subsequent corollary.

Lemma A.1. *Let y, x be real random variables (x with pdf f according to Lebesgue measure μ) and E an event of the form $s_i \in]a_i, +\infty[$, $a_i \in \mathbb{R}$, for $i \in [\![1, N]\!]$. We assume that y is conditionally independent from E given x. Then for any Y, X:*

$$\mathbb{P}(y \in Y | x \in X, E) \leqslant \sup_{x \in X} \mathbb{P}(y \in Y | x). \tag{A.3}$$

Proof. The proof follows from Bayes rule and the conditional independence between y and E:

$$\mathbb{P}(y \in Y | x \in X, E) = \frac{\mathbb{P}(y \in Y, x \in X | E)}{\mathbb{P}(x \in X | E)} = \frac{1}{\mathbb{P}(x \in X | E)} \int_{x \in X} \mathbb{P}(y \in Y | x, E) f(x) d\mu(x)$$
$$= \frac{1}{\mathbb{P}(x \in X | E)} \int_{x \in X} \mathbb{P}(y \in Y | x) f(x) d\mu(x) \leqslant \sup_{x \in X} \mathbb{P}(y \in Y | x) \frac{1}{\mathbb{P}(x \in X | E)} \int_{x \in X} f(x) d\mu(x). \tag{A.4}$$

□

Corollary A.1. *Let $y_1, \ldots, y_n, x_1, \ldots, x_n$ be real random variables and E an arbitrary random event of the form defined in Lemma A.1. We assume that for all $i \in [\![1, n]\!]$, given x_i, y_i is conditionally independent from E, x_j and y_j, for $j \neq i$. Then for any Y_i, X_i:*

$$\mathbb{P}\left(\forall i \in [\![i, n]\!], y_i \in Y_i | \forall i \in [\![1, n]\!], x_i \in X_i, E\right) \leqslant \prod_{i=1}^{n} \sup_{x \in X_i} \mathbb{P}(y_i \in Y_i | x). \tag{A.5}$$

Proof. Using Bayes rule,

$$\mathbb{P}\left(\forall i \in [\![i, n]\!], y_i \in Y_i | \forall i \in [\![1, n]\!], x_i \in X_i, E\right) \tag{A.6}$$
$$= \prod_{i=1}^{n} \mathbb{P}\left(y_i \in Y_i | \forall j \in [\![1, n]\!], x_j \in X_j, E, \forall k \in [\![1, i-1]\!], y_k \in Y_k\right) \tag{A.7}$$
$$\leqslant \prod_{i=1}^{n} \sup_{x \in X_i} \mathbb{P}(y_i \in Y_i | x) \tag{A.8}$$

where the last inequality is given by the application of Lemma A.1 on each term of the product. □

A.2.1 Energy decay in the nth step

Lemma 11.2 (Constraints propagation)**.** *Consider an assignment φ^{n+1} resulting from an iteration of Algorithm 28. Then for each pair of nodes $x, z \in \mathcal{V}$,*

$$U_x(\varphi_x^{n+1}) \geqslant \varepsilon \Rightarrow U_z(\varphi_z^{n+1}) \geqslant \varepsilon_{z,x}, \tag{A.9}$$

where the levels $\varepsilon_{z,x} \geqslant 0$ are as follows. For the ancestors of x (i.e. $\mathcal{P}(z, x) \neq \emptyset$) the levels $\varepsilon_{z,x}$ are defined via the following recursion starting from x and following the inverse propagation order:

$$\begin{cases} \varepsilon_{z,x} = \inf \left\{ U_z(\theta) \;\middle|\; \theta \in \bigcap_{y \text{ s.t. } z \sim y} A_{z,y}^{-1}(\{U_y \geqslant \varepsilon_{y,x}\}) \right\} \\ \varepsilon_{x,x} = \varepsilon. \end{cases} \tag{A.10}$$

For the rest of the nodes $\varepsilon_{z,x} = -1$.

Proof. We start by observing the following property of the merge operator. Consider a node $y \in \mathcal{V}$ and two sets of candidates patches ξ, η. Then it is easy to show that

$$|\xi| \geqslant k \quad \text{and} \quad U_y(\xi) < \varepsilon \quad \Rightarrow \quad U_y(\text{merge}_y^k(\xi \cup \eta)) < \varepsilon. \tag{A.11}$$

The proof of the lemma is trivial for the nodes that are not ancestors of x. To prove it for the ancestors we proceed by induction starting from x and following the inverse propagation order. Let z, a node in the query image $\mathbf{A}$, be an ancestor of x. We assume that the statement holds for

the nodes preceding z in the inverse propagation order (or equivalently those succeeding z with the propagation order).

In particular let y be a child of z, i.e. $z \sim y$, and assume that $\varphi_y^{n+1/2} \in \{U_y \geqslant \varepsilon_{y,x}\}$. The candidates $\varphi_y^{n+1/2}$ result from the propagation from the parents of y, among them is z. By (A.11) it follows that if $U_y(A_{z,y}\varphi_z^{n+1}) < \varepsilon_{y,x}$, then $U_y(\varphi_y^{n+1/2}) < \epsilon_{y,x}$, violating our assumption. Thus, necessarily

$$U_y(\varphi_y^{n+1/2}) \geqslant \varepsilon_{y,x} \quad \Rightarrow \quad U_z(A_{z,y}\varphi_z^{n+1}) \geqslant \varepsilon_{y,x} \quad \Rightarrow \quad \varphi_z^{n+1} \in A_{z,y}^{-1}\{U_y \geqslant \varepsilon_{y,x}\}. \tag{A.12}$$

Since this holds for all children y of z, we have that

$$\varphi_z^{n+1} \in \bigcap_{z\sim y} A_{z,y}^{-1}\{U_y \geqslant \varepsilon_{y,x}\} = L_{z,x}^{\varepsilon}. \tag{A.13}$$

Therefore

$$U_z(\varphi_z^{n+1}) \geqslant \inf U_z\left(L_{z,x}^{\varepsilon}\right) \quad \text{i.e.} \quad \varphi_z^{n+1} \in \{U_z \geqslant \varepsilon_{y,x}\}. \tag{A.14}$$

In the case of the nearest neighbor search, we can directly use the $L_{y,x}^{\varepsilon}$s instead of having to use the upper level sets defined by the $\varepsilon_{y,x}$s in the following proof for a tighter bound. □

Theorem 11.1 (Point-wise convergence)**.** *Consider the field of candidate matches at iteration n, φ^n. Define φ^{n+1} by applying an iteration of the Generic PatchMatch in Algorithm 28. Then, for all $\varepsilon > 0$, for all $x \in \boldsymbol{A}$, we have*

$$\mathbb{P}(U_x(\varphi_x^{n+1}) \geqslant \varepsilon) \leqslant \mathbb{P}(U_x(\varphi_x^{n}) \geqslant \varepsilon) \prod_{z\in\mathbf{A}} \left(C_2(z,\varepsilon_{z,x})^{\mu(z)} C_1(z,\varepsilon_{z,x})\right), \tag{A.15}$$

where $\mu(z)$ was defined in (10.5) as the number of parents of node z and C_i denotes the worst case transition probability for kernel Q_i, as in Eq. (10.8).

Proof. We consider a topological ordering of the nodes in the query image. Given two nodes $z, y \in \mathbf{A}$, we use the notation $z < y$ if z precedes y. We denote by $y-1$ and $y+1$ the nodes before and after y in the ordering.

The proof consist on a recursion on the ordered set of nodes. For $y \in \mathbf{A}$ we define the following events:

$$\begin{aligned} \mathcal{S}_y &: \quad \forall z > y, \qquad U_z(\varphi_z^n) \quad \geqslant \varepsilon_{z,x} \\ \mathcal{P}_y &: \quad \forall z \leqslant y, \quad U_z(\varphi_z^{n+1}) \quad \geqslant \varepsilon_{z,x}. \end{aligned} \tag{A.16}$$

The event $\mathcal{S}_y$ restricts the candidates at iteration n of the nodes *succeeding* y, whereas the event $\mathcal{P}_y$ considers the candidates at iteration $n+1$ of the nodes *preceding* y.

From Lemma 11.2 we have that: $U_x(\varphi_x^{n+1}) \geqslant \varepsilon \Rightarrow \forall z, U_z(\varphi_z^{n+1}) \geqslant \varepsilon_{z,x}$. Taking probabilities

$$\mathbb{P}(U_x(\varphi_x^{n+1}) \geqslant \varepsilon) \leqslant \mathbb{P}(\forall z, U_z(\varphi_z^{n+1}) \geqslant \varepsilon_{z,x}) \leqslant \mathbb{P}(\mathcal{P}_x \,;\, \mathcal{S}_x). \tag{A.17}$$

The last equality holds because for $z > x$ as the level $\varepsilon_{z,x}$ is defined as -1. Therefore the conditions over φ_z^{n+1} or φ_z^n for such nodes are trivially satisfied.

We proceed by showing that the following recursive relation holds for any node y:

$$\mathbb{P}(\mathcal{P}_y \,;\, \mathcal{S}_y) \leqslant \mathbb{P}(\mathcal{P}_{y-1} \,;\, \mathcal{S}_{y-1}) C_1(y,\varepsilon_{y,x}) C_2(y,\varepsilon_{y,x})^{\mu(y)}. \tag{A.18}$$

The result then follows by applying this recursion backwards from x until the first node in the topological ordering. To show (A.18) we note that $\mathbb{P}(\mathcal{P}_y \,;\, \mathcal{S}_y) = \mathbb{P}(\mathcal{P}_{y-1} \,;\, \mathcal{S}_y \,;\, U_y(\varphi_y^{n+1}) \geqslant$

$\varepsilon_{y,x}$). Since φ_y^{n+1} results from a propagation step 8, we have that

$$U_y(\varphi_y^{n+1}) \geq \varepsilon_{y,x} \Rightarrow U_y(merge_y^k(\varphi_y^{n+1/2} \cup S_1\varphi_y^{n+1/2})) \geqslant \varepsilon_{y,x}$$
$$\Rightarrow U_y(\varphi_y^{n+1/2}) \geqslant \varepsilon_{y,x} \text{ and } U_y(S_1\varphi_y^{n+1/2}) \geqslant \varepsilon_{y,x}. \quad \text{(A.19)}$$

Thus we have the following inequality:

$$\begin{aligned}
\mathbb{P}(\mathcal{P}_y\ ;\ \mathcal{S}_y) &\leqslant \mathbb{P}(\mathcal{P}_{y-1}\ ;\ \mathcal{S}_y\ ;\ U_y(\varphi_y^{n+1/2}) \geqslant \varepsilon_{y,x}\ ;\ U_y(S_1\varphi_y^{n+1/2}) \geqslant \varepsilon_{y,x}) \\
&\leqslant \mathbb{P}(U_y(S_1\varphi_y^{n+1/2}) \geqslant \varepsilon_{y,x} \mid \mathcal{P}_{y-1}\ ;\ \mathcal{S}_y\ ;\ U_y(\varphi_y^{n+1/2}) \geqslant \varepsilon_{y,x}) \\
&\qquad \mathbb{P}(\mathcal{P}_{y-1}\ ;\ \mathcal{S}_y\ ;\ U_y(\varphi_y^{n+1/2}) \geqslant \varepsilon_{y,x}) \\
&\leqslant \sup\{Q_{1,y}(\eta, \{U_y \geqslant \varepsilon_{y,x}\})\ :\ \eta \in \{U_y \geqslant \varepsilon_{y,x}\}\}\, \mathbb{P}(\mathcal{P}_{y-1}\ ;\ \mathcal{S}_y\ ;\ U_y(\varphi_y^{n+1/2}) \geqslant \varepsilon_{y,x}) \\
&\leqslant C_1(y, \varepsilon_{y,x})\mathbb{P}(\mathcal{P}_{y-1}\ ;\ \mathcal{S}_y\ ;\ U_y(\varphi_y^{n+1/2}) \geqslant \varepsilon_{y,x}). \qquad \text{(A.20)}
\end{aligned}$$

The third inequality comes from the application of Lemma A.1. The last step follows from the definition of C_1 in (10.8). We continue by noticing that,

$$U_y(\varphi_y^{n+1/2}) \geqslant \varepsilon_{y,x} \Rightarrow U_y(merge_y^k(\varphi_y^n \cup \bigcup_{z\sim y} A_{z,y}\varphi_z^{n+1} \cup \bigcup_{z\sim y} S_2A_{z,y}\varphi_z^{n+1})) \geqslant \varepsilon_{y,x} \Rightarrow$$
$$U_y(\varphi_y^n) \geqslant \varepsilon_{y,x}\ ;\ \forall z\sim y, U_y(A_{z,y}\varphi_z^{n+1}) \geqslant \varepsilon_{y,x}\ ;\ \forall z\sim y, U_y(S_2A_{z,y}\varphi_z^{n+1}) \geqslant \varepsilon_{y,x}. \quad \text{(A.21)}$$

To simplify the notation, in the following we will drop the subscripts from $A_{z,y}$. The implication (A.21), yields the following

$$\begin{aligned}
&\mathbb{P}(\mathcal{P}_{y-1}\ ;\ \mathcal{S}_y\ ;\ U_y(\varphi_y^{n+1/2}) \geqslant \varepsilon_{y,x}) \\
&\quad\leqslant \mathbb{P}(\mathcal{P}_{y-1}\ ;\ \mathcal{S}_y\ ;\ \forall z\sim y,\ U_y(A\varphi_z^{n+1}) \geqslant \varepsilon_{y,x},\ U_y(S_2A\varphi_z^{n+1}) \geqslant \varepsilon_{y,x}\ ;\ U_y(\varphi_y^n) \geqslant \varepsilon_{y,x}) \\
&\quad\leqslant \mathbb{P}(\mathcal{P}_{y-1}\ ;\ \mathcal{S}_{y-1}\ ;\ \forall z\sim y,\ U_y(A\varphi_z^{n+1}) \geqslant \varepsilon_{y,x},\ U_y(S_2A\varphi_z^{n+1}) \geqslant \varepsilon_{y,x}) \\
&\quad\leqslant \mathbb{P}(\forall z\sim y,\ U_y(S_2A\varphi_z^{n+1}) \geqslant \varepsilon_{y,x} \mid \mathcal{P}_{y-1}\ ;\ \mathcal{S}_{y-1}\ ;\ \forall z\sim y, U_y(A\varphi_z^{n+1}) \geqslant \varepsilon_{y,x}) \\
&\qquad\qquad \mathbb{P}(\mathcal{P}_{y-1}\ ;\ \mathcal{S}_{y-1}\ ;\ \forall z\sim y\ ,\ U_y(A\varphi_z^{n+1}) \geqslant \varepsilon_{y,x}) \\
&\quad\leqslant \prod_{z\sim y} \sup\{Q_{2,y}(\eta, \{U_y \geqslant \varepsilon_{y,x}\})\ :\ \eta \in \{U_y \geqslant \varepsilon_{y,x}\}\} \\
&\qquad\qquad \mathbb{P}(\mathcal{P}_{y-1}\ ;\ \mathcal{S}_{y-1}\ ;\ \forall z\sim y\ ,\ U_y(A\varphi_z^{n+1}) \geqslant \varepsilon_{y,x}) \\
&\quad\leqslant C_2(y, \{U_y \geqslant \varepsilon_{y,x}\})^{\mu(x)}\mathbb{P}(\mathcal{P}_{y-1}\ ;\ \mathcal{S}_{y-1}\ ;\ \forall z\sim y\ ,\ U_y(A\varphi_z^{n+1}) \geqslant \varepsilon_{y,x}) \\
&\quad\leqslant C_2(y, \{U_y \geqslant \varepsilon_{y,x}\})^{\mu(x)}\mathbb{P}(\mathcal{P}_{y-1}\ ;\ \mathcal{S}_{y-1}). \qquad \text{(A.22)}
\end{aligned}$$

In the third step we have applied Corollary A.1, whereas the forth step results from the definition of C_2.

The recursion (A.18) follows from (A.20) and (A.22). □

Remark A.1. *The proof is based on the fact that, due to the propagation, restricting the energy of the candidates at x implies constraints on the candidates of its ancestors (Lemma 11.2). We then bound the probability of all random samples drawn to satisfy those constraints. The more restrictive those constraints are, the smaller the probability of satisfying them. The Lemma in 11.2 establishes that $\varphi_z^{n+1} \in \{U_z \geqslant \varepsilon_{z,x}\}$. However, during the proof of Lemma 11.2 it is shown that the candidate sets φ_z^{n+1} belong to a set $L_{z,x}^{\varepsilon}$, which can be smaller than $\{U_z \geqslant \varepsilon_{z,x}\}$ (i.e. $L_{z,x}^{\varepsilon} \subseteq \{U_z \geqslant \varepsilon_{z,x}\}$. A tighter bound can be derived by considering this more restrictive constraint, and bounding the probability of $S_1\varphi_z^{n+1/2} \in L_{z,x}^{\varepsilon}$. In the cases of k-sets, the set $L_{z,x}^{\varepsilon}$ is a set of k-sets, and it is difficult to evaluate the probability of sampling a k-set in that set. This difficulty disappears for $k = 1$. Then $L_{z,x}^{\varepsilon} \subseteq \boldsymbol{B}$, and computing the probability of sampling in the allowed set becomes easier. Thus a tighter bound can be computed for $k = 1$.*

Theorem 11.2. *Consider the field of candidate matches at iteration n, φ^n. Define φ^{n+1} by applying an iteration of the Generic PatchMatch in Algorithm 28. Then, for all $\varepsilon > 0$ we have*

$$\mathbb{P}(\|U_\cdot(\varphi_\cdot^{n+1})\|_\infty \geqslant \varepsilon) \leqslant \mathbb{P}(\|U_\cdot(\varphi_\cdot^{n})\|_\infty \geqslant \varepsilon) \prod_{z \in \mathbf{A}} \left(C_2(z,\varepsilon)^{\mu(z)} C_1(z,\varepsilon) \right). \tag{A.23}$$

Proof. The proof is similar to the one of Theorem 11.1 where instead of using Lemma 11.2 we use $\|U_\cdot(\varphi_\cdot^{n+1})\|_\infty \geqslant \varepsilon$, i.e. $\forall x \in \mathbf{A}, U_x(\varphi_x^{n+1}) \geqslant \varepsilon$. □

Corollary 11.1. *Assume that for any pair (η, ξ) of sets of k candidate matches $Q_1(\eta,\xi) > 0$ (or $Q_2(\eta,\xi) > 0$). Let (φ^n) be a sequence defined by Algorithm 28. Then $\forall x \in \boldsymbol{A}, \mathbb{E}[U_x(\varphi_x^n)] \xrightarrow[n\to\infty]{} 0$ and $\mathbb{E}[\|U_\cdot(\varphi_\cdot^n)\|_\infty] \xrightarrow[n\to\infty]{} 0$.*

Proof. We will show the convergence in the mean for a single node $x \in \mathbf{A}$, i.e. $\mathbb{E}[|U_x(\varphi_x^n)|] \xrightarrow[n\to\infty]{} 0$. We recall that for a non-negative random variable X, $\mathbb{E}[X] = \int_{x>0} \mathbb{P}(X \geqslant x)$. Then we have:

$$\begin{aligned}
\mathbb{E}[|U_x(\varphi_x^{n+1})|] = \mathbb{E}[U_x(\varphi_x^{n+1})] &= \int_{\varepsilon>0} \mathbb{P}(U_x(\varphi_x^{n+1}) \geqslant \varepsilon) && \text{(A.24)}\\
&\leqslant \int_{\varepsilon>0} \prod_{z\in\mathbf{A}} \left(C_2(z,\varepsilon_{z,x})^{\mu(z)} C_1(z,\varepsilon_{z,x}) \right) \mathbb{P}(U_x(\varphi_x^n) \geqslant \varepsilon) && \text{(A.25)}\\
&\leqslant \int_{\varepsilon>0} C_2(x,\varepsilon)^{\mu(x)} C_1(x,\varepsilon)\, \mathbb{P}(U_x(\varphi_x^n) \geqslant \varepsilon) && \text{(A.26)}\\
&\leqslant \int_{\varepsilon>0} \left(\sup_{\alpha>0} C_2(x,\alpha) \right)^{\mu(x)} \left(\sup_{\alpha>0} C_1(x,\alpha) \right) \mathbb{P}(U_x(\varphi_x^n) \geqslant \varepsilon) && \text{(A.27)}\\
&\leqslant \left(\sup_{\alpha>0} C_2(x,\alpha) \right)^{\mu(x)} \left(\sup_{\alpha>0} C_1(x,\alpha) \right) \int_{\varepsilon>0} \mathbb{P}(U_x(\varphi_x^n) \geqslant \varepsilon) && \text{(A.28)}\\
&\leqslant \left(\sup_{\alpha>0} C_2(x,\alpha) \right)^{\mu(x)} \left(\sup_{\alpha>0} C_1(x,\alpha) \right) \mathbb{E}[U_x(\varphi_x^n)]. && \text{(A.29)}
\end{aligned}$$

Since $Q_1(\eta,\xi) > 0$, for all η, ξ, we have that $Q_1(\eta, \{U_x = 0\}) > 0$ for any η. In the discrete case (the one only one considered for this theorem) the sup is achieved and is not 1. Therefore $\sup_{\alpha>0} C_1(x,\alpha) = \max_{\alpha>0} C_1(x,\alpha) < 1$ and the convergence follows. With a similar derivation can be used to show the convergence of the L_∞ norm of the whole NNF. □

A.3 Specific *PatchMatch* algorithms

A.3.1 The original *PatchMatch* algorithm

Proposition 11.1. *The specific basic PatchMatch algorithm described in this section algorithm converges in probability to a NNF which minimizes the energy, namely*

$$\lim_{n\to\infty} \mathbb{P}(U_x(\varphi_x^n) \geqslant \varepsilon) = 0, \forall \varepsilon > 0, x \in \boldsymbol{A}, \tag{A.30}$$

with a geometric convergence rate.
Moreover for all $\varepsilon > 0$, for all $x \in \boldsymbol{A}$, we have that

$$\mathbb{P}(U_x(\varphi_x^{n+1}) \geqslant \varepsilon) \leqslant \mathbb{P}(U_x(\varphi_x^n) \geqslant \varepsilon) \prod_{z\in\mathbf{A}} \left(1 - (1 - C'(c_i, \varepsilon_{z,x}))^k \right), \tag{A.31}$$

with

$$C'(z,\alpha) := \sup_\eta Q'(\eta, \{U_z \geqslant a\}). \tag{A.32}$$

For $\alpha > 0$ we can guarantee that $C'(z,\alpha) < 1$.

Proof. The bound in Theorem 11.1 applies. We now express C_1 in terms of the new C'. For that, we compute an upper bound for $\mathbb{P}(U_z(S_1\varphi_z^n) \geqslant a \mid U_z(\varphi_z^n) \geqslant a, E)$ with $a \geqslant 0$ and E any event, $\forall z \in \mathbf{A}$. We first remark that, if $S_1^i\varphi$ is the i^{th} sample generated from the random sampling around φ,

$$\mathbb{P}(U_z(S_1\varphi_z^n) \geqslant a \mid U_z(\varphi_z^n) \geqslant a, E) = 1 - \prod_{l=1}^{k} \mathbb{P}(U_z(S_1^l\varphi_z^n) \leqslant a \mid U_z(\varphi_z^n) \geqslant a, E). \quad \text{(A.33)}$$

For a candidate $\phi \in S_1\varphi_z^n$,

$$\mathbb{P}(U_z(\phi) \geqslant a \mid U_z(\varphi_z^n) \geqslant a, E) \leqslant \sup_{\eta \in \{U_z \geqslant a\}} Q(\eta, \{U_z \geqslant a\}) \quad \text{(A.34)}$$

Let us remind that η and $S\eta$ are k-sets, i.e. sets of k distinct candidates. The candidates in $S\eta$ are sampled centered at the best candidate in η (the one minimizing U_z). We know that $\eta \in \{U_z \geqslant a\}$, which only constrains the worst candidate in η, but says nothing about the best candidate. This is why for this proof there is no restriction on where the sample comes from. Therefore, with

$$\sup_{\eta} Q'(\eta, \{U_z \geqslant a\}) =: C'(z, a) \quad \text{(A.35)}$$

we have

$$\mathbb{P}(U_z(\phi) \leqslant a \mid U_z(\varphi_z^n) \geqslant a, E) \geqslant 1 - C'(z, a) \quad \text{(A.36)}$$

and

$$\mathbb{P}(U_z(S_1\varphi_z^n) \geqslant a \mid U_z(\varphi_z^n) \geqslant a, E) \leqslant 1 - (1 - C'(z, a))^k. \quad \text{(A.37)}$$

Since the support of the random search is the full image, it guarantees that $\forall z \in \mathbf{A}, \forall a > 0, C'(z, a) < 1$. This implies that the bound found is strictly inferior to one and therefore the convergence is insured. □

Corollary 11.2. *In the case of the search of the nearest neighbor, the upper bound can be written as*

$$\mathbb{P}(U_x(\varphi_x^{n+1}) \geqslant \varepsilon) \leqslant \prod_{z \in \boldsymbol{A}} C'(z, \varepsilon_{z,x}) \mathbb{P}(U_x(\varphi_x^n) \geqslant \varepsilon). \quad \text{(A.38)}$$

with

$$C'(z, \alpha) := \sup_{\eta \in \{U_z \geqslant a\}} Q'(\eta, \{U_z \geqslant a\}). \quad \text{(A.39)}$$

This bound is actually tighter than the one derived in [ACF12].

Proof. The derivation of the tighter bound presented in Equation (A.38) is the same as in Proposition 11.1. The difference comes from the that in the case $k = 1$, the best current match is also the worst current match therefore the element used to sample is necessarily with an energy larger than $\varepsilon_{z,x}$.

The second part of the proof concerns the comparison with the bound of [ACF12]. The difference between the bound (A.38) and the one in [ACF12] lies in the levels $\varepsilon_{z,x}$ inside the factors C'. The levels used in [ACF12] are of the form $\varepsilon - \ell_{z,x}$, where $\ell_{z,x}$ is defined as follows:

$$\ell_{z,x} = \min_{c \in \mathcal{P}(z,x)} \sum_{i=1}^{n} d_{c_i, c_{i-1}} \quad \text{(A.40)}$$

with $\mathcal{P}(z, x)$ being the set of all the paths from z to x in the graph and $d_{c_i,c_{i-1}} = \|U_{c_i} - U_{c_{i-1}} \circ A\|_\infty$ Therefore, due to the monotonicity of C' we just have to show that $\varepsilon_{z,x} \geqslant \varepsilon - \ell_{z,x}$.

For $z = x$, $\varepsilon_{z,x} = \varepsilon$ and $\ell_{z,x} = 0$ therefore the property is verified. Suppose now that the property is true for any y such that $y > z$. We will show that in this case the property stays true for z. Suppose that $\varepsilon_{z,x} < \varepsilon - \ell_{z,x}$. Because $\varepsilon_{z,x} = \inf U_z(L^{\varepsilon}_{z,x})$, this means that it exists $\eta \in L^{\varepsilon}_{z,x}$ such that $U_z(\eta) < \varepsilon - \ell_{z,x}$. Let y be the child of z such that $\ell_{z,x} = \ell_{y,x} + d_{z,y}$. In this case we have that

$$U_z(\eta) + d_{z,y} < \varepsilon - \ell_{z,x} + d_{z,y} \tag{A.41}$$

therefore

$$U_y(A\eta) \leqslant U_z(\eta) + d_{z,y} < \varepsilon - \ell_{y,x}. \tag{A.42}$$

We then have found $\eta \in L^{\varepsilon}_{y,x}$ *i.e.* $A\eta \in \{U_y \geqslant \varepsilon_{y,x}\}$ therefore $\varepsilon_{y,x} \leqslant U_y(A\eta) < \varepsilon - \ell_{y,x}$ which is contradictory with the hypothesis. The property is then also valid for z. □

Remark A.2. *The tighter bound for the specific case of $k = 1$ mentioned in the Remark after the proof of Theorem A.15 applies in this case as well, yielding a bound for the original PatchMatch that is tighter than the one from Proposition 11.1. We compared both bounds and in practice the difference between them is small. For our experiments in the paper we have used a bound similar to the one of the Proposition 11.1.*

A.3.2 The CSH algorithm

Lemma 11.3. *If $\mathcal{H}$ is (R, cR, p_1, p_2)-sensitive then an OR family $\mathcal{G}$ created using n functions from $\mathcal{H}$ is $(R, cR, 1-(1-p_1)^n, 1-(1-p_2)^n)$-sensitive.*

Proof. Let p, q such that $\|p - q\| \leqslant R$ and $g \in \mathcal{G}$ generated by $h_1, \ldots, h_n \in \mathcal{H}$ with $\mathcal{H}$ (R, cR, p_1, p_2)-sensitive.

$$\mathbb{P}_{\mathcal{G}}(g(p) \neq g(q)) = \mathbb{P}_{\mathcal{H}}(h_1(p) \neq h_1(q), \ldots, h_n(p) \neq h_n(q)) \tag{A.43}$$

Because the h_is are independent,

$$\mathbb{P}_{\mathcal{H}}(h_1(p) \neq h_1(q), \ldots, h_n(p) \neq h_n(q)) = \prod_{i=1}^{n} \mathbb{P}_{\mathcal{H}}(h_i(p) \neq h_i(q)) \tag{A.44}$$

Using the (R, cR, p_1, p_2)-sensitive property of $\mathcal{H}$, for all i

$$\mathbb{P}_{\mathcal{H}}(h_i(p) \neq h_i(q)) = 1 - \mathbb{P}_{\mathcal{H}}(h_i(p) = h_i(q)) \tag{A.45}$$

$$\leqslant 1 - p_1 \tag{A.46}$$

Therefore

$$\mathbb{P}_{\mathcal{G}}(g(p) \neq g(q)) \leqslant (1 - p_1)^n \tag{A.47}$$

and

$$\mathbb{P}_{\mathcal{G}}(g(p) = g(q)) \geqslant 1 - (1 - p_1)^n. \tag{A.48}$$

Using similar derivation, the corresponding result can be proved for the second part of the sensitive definition. This result in $\mathcal{G}$ an OR family function being $(R, cR, 1-(1-p_1)^n, 1-(1-p_2)^n)$-sensitive. □

Proposition 11.2. *For a (R, cR, p_1, p_2)-sensitive family of hashing functions such that $R \geqslant \max_{z \in \boldsymbol{A}} K_z$ (see Definition 10.1), the sequence (φ^n) defined by the CSH algorithm converges in probability to a minimizer of the total energy, in the sense that*

$$\lim_{n \to \infty} \mathbb{P}(U_x(\varphi^n_x) \geqslant \varepsilon) = 0, \forall \varepsilon > 0, x \in \boldsymbol{A}, \tag{A.49}$$

with a geometric convergence rate. Moreover for all $\varepsilon > 0$, for all $x \in \boldsymbol{A}$,

$$\mathbb{P}(U_x(\varphi_x^{n+1}) \geqslant \varepsilon) \leqslant \mathbb{P}(U_x(\varphi_x^n) \geqslant \varepsilon) \prod_{z \in \mathbf{A}} C_2(z, \{U_z \geqslant \varepsilon_{z,x}\})^{\mu(z)} f(p_1, \varepsilon_{z,x}\}), \tag{A.50}$$

where $f(\alpha, \beta) = (1 - \alpha^k)$ if $\varepsilon_{z,x} > 0$, 1 otherwise.

From a set of LSH hash functions $\mathcal{H}$, a OR family of function $\mathcal{G}$ can also be defined. The function $g \in \mathcal{G}$ is based on a set of n random functions $h_1, \ldots, h_n$ from $\mathcal{H}$ such that for all p, q, $g(p) = g(q)$ if and only if there exist $i \in [\![1, n]\!]$ such that $h_i(p) = h_i(q)$.

Proof. Let $R \geqslant \max_{z \in \mathbf{A}} K_z$ and $\mathcal{H}$ an (R, cR, p_1, p_2)-sensitive family of functions. Consider also $\mathcal{G}$ an OR family function based on $\mathcal{H}$ so that a function from $\mathcal{G}$ is generated using at least k functions from $\mathcal{H}$. An upper bound for $\mathbb{P}(U_x(S_1\varphi_x^n) \geqslant a \mid E)$ with $a > 0$, $\forall x \in \mathbf{A}$ and E an undefined event will now be derived.
Firstly,

$$\forall p \in \mathbf{B}, \|p - x\| \leqslant R \Rightarrow \mathbb{P}(h(p) = h(x)) \geqslant p_1. \tag{A.51}$$

R is then chosen such that for all $x \in \mathbf{A}$, $\|x - N_k(x)\| \leqslant R$, where $N_k(x) \in \mathbf{B}$ is the k^{th} nearest neighbor of x. This property is true for the l^{th} nearest neighbor $N_l(x)$, with $l \leqslant k$. Therefore we have $\mathbb{P}(h(N_l(x)) = h(x)) \geqslant p_1$. Moreover the binning is independent from the current matching φ_x^n.

For a given $h \in \mathcal{H}$ and $\mathcal{H}' \subset \mathcal{H}$ such that $h \in \mathcal{H}'$, for a query q, for all $p \in \mathbf{B}$

$$h(q) = h(p) \Rightarrow \exists h' \in \mathcal{H}', h'(q) = h'(p) \tag{A.52}$$

Therefore for a list $H' = \{h_1, \ldots, h_k\} \subset \mathcal{H}$, for a query q, for all $p \in \mathbf{B}$

$$h_1(q) = h_1(p_1), \ldots, h_k(q) = h_k(p_n) \Rightarrow (\exists h \in \mathcal{H}', h(q) = h(p_1)), \ldots, (\exists h \in \mathcal{H}', h(q) = h(p_k)) \tag{A.53}$$

and

$$\begin{aligned}\mathbb{P}(h_1(q) = h_1(p_1), \ldots, h_k(q) = h_k(p_n)) \\ \leqslant \mathbb{P}((\exists h \in \mathcal{H}', h(q) = h(p_1)), \ldots, (\exists h \in \mathcal{H}', h(q) = h(p_k)))\end{aligned} \tag{A.54}$$

When working with the elements of the OR family $\mathcal{G}$, we can define $\mathcal{H}'$ as the set functions from $\mathcal{H}$ used to generate g ($\mathcal{H}'$ verifies the properties of the previously used $\mathcal{H}'$ because at least k functions are used to generate each element of $\mathcal{G}$),

$$\begin{aligned}\mathbb{P}_{\mathcal{G}}(g(q) = g(p_1), \ldots, g(q) = g(p_n)) \\ = \mathbb{P}_{\mathcal{H}}((\exists h \in \mathcal{H}', h(q) = h(p_1)), \ldots, (\exists h \in \mathcal{H}', h(q) = h(p_n))) \\ \geqslant \mathbb{P}_{\mathcal{H}}(h_1(q) = h_1(p_1), \ldots, h_n(q) = h_n(p_n))\end{aligned} \tag{A.55}$$

which leads to

$$\mathbb{P}_{\mathcal{G}}(g(q) = g(p_1), \ldots, g(q) = g(p_k)) \geqslant \mathbb{P}_{\mathcal{H}}(h_1(q) = h_1(p_1), \ldots, h_k(q) = h_k(p_k)) \tag{A.56}$$

because the h_i are independent

$$\mathbb{P}_{\mathcal{G}}(g(q) = g(p_1), \ldots, g(q) = g(p_n)) \geqslant \prod_{i=1}^{k} \mathbb{P}_{\mathcal{H}}(h_i(q) = h_i(p_i)) \geqslant p_1^k \tag{A.57}$$

when using the original LSH property for $\mathcal{H}$. This implies that

$$\mathbb{P}(U_x(S_1\varphi_x^n) = 0) = \mathbb{P}_{\mathcal{G}}(g(q) = g(p_1), \ldots, g(q) = g(p_n)) \geqslant p_1^k \tag{A.58}$$

Finally, $\mathbb{P}(U_x(S_1\varphi_x^n) \geqslant a \mid E) \leqslant 1 - \mathbb{P}(U_x(S_1\varphi_x^n) \leqslant a \mid E) \leqslant 1 - \mathbb{P}(U_x(S_1\varphi_x^n) = 0 \mid E) \leqslant 1 - p_1^k$ Replacing $C_1(z, \varepsilon)$, for $z \in \mathbf{A}$ and $\varepsilon > 0$, by $(1 - p_1^k)$ in the proof of Theorem 10.1 gives the result in Proposition 10.2. □

Bibliography

[ABC+10] I. Amerini, L. Ballan, R. Caldelli, A. Del Bimbo, and G. Serra. Geometric tampering estimation by means of a sift-based forensic analysis. In *ICASSP*, 2010.

[ABM10] E. Ardizzone, A. Bruno, and G. Mazzola. Detecting multiple copies in tampered images. In *ICIP*. IEEE, 2010.

[ACF12] Pablo Arias, Vicent Caselles, and Gabriele Facciolo. Analysis of a variational framework for exemplar-based image inpainting. *Multiscale Modeling & Simulation*, 10(2):473–514, 2012.

[AEHOR15] Amir Adler, Michael Elad, Yacov Hel-Or, and Ehud Rivlin. Sparse coding with anomaly detection. *Journal of Signal Processing Systems*, 79(2):179–188, 2015.

[AFCS11] Pablo Arias, Gabriele Facciolo, Vicent Caselles, and Guillermo Sapiro. A variational framework for exemplar-based image inpainting. *International Journal of Computer Vision*, 93(3):319–347, 2011.

[AFM18a] P. Arias, G. Facciolo, and J.-M. Morel. A comparison of patch-based models in video denoising. In *2018 IEEE 13th Image, Video, and Multidimensional Signal Processing Workshop (IVMSP)*, pages 1–5, June 2018.

[AFM18b] Pablo Arias, Gabriele Facciolo, and Jean-Michel Morel. A comparison of patch-based models in video denoising. In *IEEE Image, Video, and Multidimensional Signal Processing Workshop*, pages 1–5. IEEE, 2018.

[AH17] Redha A Ali and Russell C Hardie. Recursive non-local means filter for video denoising. *EURASIP Journal on Image and Video Processing*, 2017.

[AI06] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 459–468. IEEE, 2006.

[AINR14] Alexandr Andoni, Piotr Indyk, Huy L Nguyen, and Ilya Razenshteyn. Beyond locality-sensitive hashing. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1018–1028. SIAM, 2014.

[ALB18] Abdelrahman Abdelhamed, Stephen Lin, and Michael S. Brown. A high-quality denoising dataset for smartphone cameras. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[AM15] Pablo Arias and Jean-Michel Morel. Towards a bayesian video denoising method. In *Advanced Concepts for Intelligent Vision Systems*, LNCS. Springer, 2015.

[AM18a] Pablo Arias and Jean-Michel Morel. Video denoising via empirical bayesian estimation of space-time patches. *Journal of Mathematical Imaging and Vision*, 60(1):70–93, Jan 2018.

[AM18b] Pablo Arias and Jean-Michel Morel. Video denoising via empirical bayesian estimation of space-time patches. *Journal of Mathematical Imaging and Vision*, 60(1):70–93, 2018.

[AM19a] P. Arias and J.-M. Morel. Kalman filtering of patches for frame recursive video denoising. In *2019 IEEE Conf. on Comp. Vision and Pattern Recognition (CVPR) Workshops (NTIRE)*, 2019.

[AM19b] Pablo Arias and Jean-Michel Morel. Kalman Filtering of Patches for Frame-Recursive Video Denoising. In *The IEEE Conference on Computer Vision and Pattern Recognition Workshops*, June 2019.

[AMD15] AMD. *AMD APP SDK OpenCL^{TM} Optimization Guide*, 2015.

[An16] Jinwon An. Variational Autoencoder based Anomaly Detection using Reconstruction Probability. *CoRR*, 2016.

[Ans48] Francis J Anscombe. The transformation of poisson, binomial and negative-binomial data. *Biometrika*, 35(3/4):246–254, 1948.

[AT10] Dror Aiger and Hugues Talbot. The phase only transform for unsupervised surface defect detection. In *CVPR*, 2010.

[ATO15] Hiroki Akiyama, Masayuki Tanaka, and Masatoshi Okutomi. Pseudo four-channel image denoising for noisy cfa raw data. In *2015 IEEE International Conference on Image Processing (ICIP)*, pages 4778–4782. IEEE, 2015.

[Bar09] A. Barbu. Training an active random field for real-time image denoising. *IEEE Transactions on Image Processing*, 18(11):2451–2462, Nov 2009.

[BCF16] E. Bik, A. Casadevall, and F. Fang. The prevalence of inappropriate image duplication in biomedical research publications. *MBio*, 7(3):e00809–16, 2016.

[BCM05a] Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. A non-local algorithm for image denoising. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2(0):60–65, 2005.

[BCM05b] Antoni Buades, Bartomeu Coll, and Jean Michel Morel. Denoising image sequences does not require motion estimation. In *Proceedings of the IEEE Conference on Advanced Video and Signal Based Surveillance*, pages 70–74, 2005.

[BCM05c] Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. A non-local algorithm for image denoising. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 60–65. IEEE, 2005.

[BCM08] Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. Nonlocal image and movie denoising. *International Journal of Computer Vision*, 76(2):123–139, 2008.

[BCM11] Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. Non-local means denoising. *IPOL*, 1:208–212, 2011.

[BCW14] Giacomo Boracchi, Diego Carrera, and Brendt Wohlberg. Novelty detection in images by sparse representations. In *Intelligent Embedded Systems (IES)*, 2014.

[BD19] Thibaud Briand and Axel Davy. Optimization of Image B-spline Interpolation for GPU Architectures. *IPOL*, 9:183–204, 2019.

[Ben75] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[BFS18] Thibaud Briand, Gabriele Facciolo, and Javier Sánchez. Improvements of the Inverse Compositional Algorithm for Parametric Motion Estimation. *Image Processing On Line*, 8:435–464, 2018.

[BGvGM18] Q. Bammey, R. Grompone von Gioi, and J.-M. Morel. Automatic detection of demosaicing image artifacts and its use in tampering detection. In *IEEE MIPR*, pages 424–429. IEEE, 2018.

[BI07] Oren Boiman and Michal Irani. Detecting irregularities in images and in video. *International Journal of Computer Vision*, 74(1):17–31, 2007.

[BI12] Ali Borji and Laurent Itti. Exploiting local and global patch rarities for saliency detection. In *CVPR*, 2012.

[BJGY10] X. Bo, W. Junwen, L. Guangjie, and D. Yuewei. Image copy-move forgery detection based on surf. In *MINES*, pages 889–892. IEEE, 2010.

[BK95] James C. Brailean and Aggelos K. Katsaggelos. Simultaneous recursive displacement estimation and restoration of noisy-blurred image sequences. *IEEE Transactions on Image Processing*, 4(9):1236–1251, 1995.

[BKB+09] Jérôme Boulanger, Charles Kervrann, Patrick Bouthemy, Peter Elbau, Jean-Baptiste Sibarita, and Jean Salamero. Patch-based nonlocal functional for denoising fluorescence microscopy image sequences. *IEEE TMI*, 29(2):442–454, 2009.

[BKC08] Thomas Brox, Oliver Kleinschmidt, and Daniel Cremers. Efficient nonlocal means for denoising of textural patterns. *IEEE TIP*, 17(7):1083–1092, 2008.

[BL17] A. Buades and J. L. Lisani. Dual domain video denoising with optical flow estimation. In *IEEE ICIP*, Sep. 2017.

[BLM16] Antoni Buades, Jose-Luis Lisani, and Marko Miladinović. Patch-based video denoising with optical flow estimation. *IEEE Transactions on Image Processing*, 25(6):2573–2586, June 2016.

[BM01] Simon Baker and Iain Matthews. Equivalence and efficiency of image alignment algorithms. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages I–1090. Citeseer, 2001.

[BR14] Giacomo Boracchi and Manuel Roveri. Exploiting self-similarity for change detection. In *2014 International Joint Conference on Neural Networks*, pages 3339–3346. IEEE, 2014.

[BR19] Joshua Batson and Loic Royer. Noise2Self: Blind Denoising by Self-Supervision. In *The International Conference on Machine Learning (ICML)*, 2019.

[Bra00] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[BRR11] Michael Bleyer, Christoph Rhemann, and Carsten Rother. Patchmatch stereo - stereo matching with slanted support windows. In *Proceedings of the British Machine Vision Conference (BMVA)*, pages 14.1–14.11. BMVA Press, 2011.

[BSFG09a] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan Goldman. Patchmatch: a randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics-TOG*, 28(3):24, 2009.

[BSFG09b] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. In *ACM SIGGRAPH 2009 Papers*. Association for Computing Machinery, 2009.

[BSGF10] Connelly Barnes, Eli Shechtman, Dan B Goldman, and Adam Finkelstein. The generalized patchmatch correspondence algorithm. In *Proceedings of the Europ. Conf. on Computer Vision (ECCV)*, pages 29–43. Springer, 2010.

[BSH12] H. C. Burger, C. J. Schuler, and S. Harmeling. Image denoising: Can plain neural networks compete with bm3d? In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2392–2399, June 2012.

[BSM09] S. Bayram, H. Sencar, and N. Memon. An efficient and robust method for detecting copy-move forgery. In *ICASSP*, pages 1053–1056. IEEE, 2009.

[BT06] Neil Bruce and John Tsotsos. Saliency based on information maximization. In *NIPS*, 2006.

[BYJ14] Linchao Bao, Qingxiong Yang, and Hailin Jin. Fast edge-preserving patchmatch for large displacement optical flow. *Image Processing, IEEE Transactions on*, 23(12):4996–5006, Dec 2014.

[BZL+15] Connelly Barnes, Fang-Lue Zhang, Liming Lou, Xian Wu, and Shi-Min Hu. Patchtable: Efficient patch queries for large datasets and applications. In *ACM Transactions on Graphics (Proc. SIGGRAPH)*, August 2015.

[BZZM15] N. Ben-Zrihem and L. Zelnik-Manor. RIANN: Approximate Nearest Neighbor Fields in Video. In *Proceedings of the IEEE Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[CBFW15] Diego Carrera, Giacomo Boracchi, Alessandro Foi, and Brendt Wohlberg. Detecting anomalous structures by convolutional sparse models. In *IJCNN*, 2015.

[CBK09] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[CCC+10] Antonin Chambolle, Vicent Caselles, Daniel Cremers, Matteo Novaga, and Thomas Pock. An introduction to total variation for image analysis. *Theoretical foundations and numerical methods for sparse recovery*, 9(263-340):227, 2010.

[CCCY18] Jingwen Chen, Jiawei Chen, Hongyang Chao, and Ming Yang. Image blind denoising with generative adversarial network based noise modeling. In *CVPR*, pages 3155–3164. IEEE, 2018.

[CCXK18] Chen Chen, Qifeng Chen, Jia Xu, and Vladlen Koltun. Learning to see in the dark. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[CFGS12] Xiaobai Chen, Tom Funkhouser, Dan B. Goldman, and Eli Shechtman. Non-parametric texture transfer using meshmatch. Technical Report Technical Report 2012-2, Adobe, November 2012.

[CFKE18] C. Cruz, A. Foi, V. Katkovnik, and K. Egiazarian. Nonlocality-reinforced convolutional neural networks for image denoising. *IEEE Signal Processing Letters*, 25(8):1216–1220, Aug 2018.

[CGFY12] Y. Cao, T. Gao, L. Fan, and Q. Yang. A robust detection algorithm for copy-move forgery in digital images. *Forensic science international*, 214(1-3):33–43, 2012.

[CHKB09] Pierrick Coupé, Pierre Hellier, Charles Kervrann, and Christian Barillot. Nonlocal means-based speckle filtering for ultrasound images. *IEEE TIP*, 18(10):2221–2229, 2009.

[CL97] Antonin Chambolle and Pierre-Louis Lions. Image recovery via total variation minimization and related problems. *Numerische Mathematik*, 76(2):167–188, 1997.

[CM12] Laurent Condat and Saleh Mosaddegh. Joint demosaicking and denoising by total variation minimization. In *2012 19th IEEE International Conference on Image Processing*, pages 2781–2784. IEEE, 2012.

[CMPT+17] Sergi Caelles, Kevis-Kokitsi Maninis, Jordi Pont-Tuset, Laura Leal-Taixé, Daniel Cremers, and Luc Van Gool. One-shot video object segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 221–230, 2017.

[Col14] Miguel Colom. *Multiscale noise estimation and removal for digital images*. PhD thesis, Universitat de les Illes Balears, 2014.

[CP17] Yunjin Chen and Thomas Pock. Trainable Nonlinear Reaction Diffusion: A Flexible Framework for Fast and Effective Image Restoration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1256–1272, 6 2017.

[CPV15] Davide Cozzolino, Giovanni Poggi, and Luisa Verdoliva. Efficient dense-field copy–move forgery detection. *IEEE Transactions on Information Forensics and Security*, 10(11):2284–2297, 2015.

[CRJ+12] Vincent Christlein, Christian Riess, Johannes Jordan, Corinna Riess, and Elli Angelopoulou. An evaluation of popular copy-move forgery detection approaches. *IEEE Transactions on information forensics and security*, 7(6):1841–1854, 2012.

[CSY16] Xinyuan Chen, Li Song, and Xiaokang Yang. Deep rnns for video denoising. In *Applications of Digital Image Processing*, 2016.

[CvG19] Michele Claus and Jan van Gemert. Videnn: Deep blind video denoising. In *IEEE CVPRW*, 2019.

[CYB06] Pierrick Coupé, Pierre Yger, and Christian Barillot. Fast non local means denoising for 3d mr images. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 33–40. Springer, 2006.

[DAG10] Vincent Duval, Jean-François Aujol, and Yann Gousseau. On the parameter choice for the non-local means. 2010.

[DDS+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[DDT12] Charles-Alban Deledalle, Loïc Denis, and Florence Tupin. How to compare noisy patches? Patch similarity beyond Gaussian noise. *International Journal of Computer Vision*, 99(1):86–102, 2012.

[DDT+14] Charles-Alban Deledalle, Loïc Denis, Florence Tupin, Andreas Reigber, and Marc Jäger. Nl-sar: A unified nonlocal framework for resolution-preserving (pol)(in) sar denoising. *IEEE Transactions on Geoscience and Remote Sensing*, 53(4):2021–2038, 2014.

[DE] Axel Davy and Thibaud Ehret. Gpu acceleration of nl-means, bm3d and vbm3d. *Journal of Real-Time Image Processing*, pages 1–18.

[DE20] Axel Davy and Thibaud Ehret. GPU acceleration of NL-means, BM3D and VBM3D. *Journal of Real-Time Image Processing*, pages 1–18, 2020.

[DEF+18] Axel Davy, Thibaud Ehret, Gabriele Facciolo, Jean-Michel Morel, and Pablo Arias. Non-local video denoising by cnn. *arXiv preprint arXiv:1811.12758*, 2018.

[DEM+19] Axel Davy, Thibaud Ehret, Jean-Michel Morel, Pablo Aria, and Gabriele Facciolo. A non-local cnn for video denoising. In *IEEE ICIP*, 2019.

[DEM+20] Axel Davy, Thibaud Ehret, Jean-Michel Morel, Pablo Arias, and Gabriele Facciolo. Video denoising by combining patch search and cnns. *Journal of Mathematical Imaging and Vision*, pages 1–16, 2020.

[DEMD18] Axel Davy, Thibaud Ehret, Jean-Michel Morel, and Mauricio Delbracio. Reducing anomaly detection in images to detection in noise. In *IEEE International Conference on Image Processing (ICIP)*, pages 1058–1062. IEEE, 2018.

[DF06] Kostadin Dabov and Alessandro Foi. Image denoising with block-matching and 3D filtering. *Electronic Imaging*, 6064:1–12, 2006.

[DFBCH11] Fernanda Palhano Xavier De Fontes, Guillermo Andrade Barroso, Pierrick Coupé, and Pierre Hellier. Real time ultrasound image denoising. *Journal of real-time image processing*, 6(1):15–22, 2011.

[DFE07] Kostadin Dabov, Alessandro Foi, and Karen Egiazarian. Video denoising by sparse 3D transform-domain collaborative filtering. In *Proc. 15th European Signal Processing Conference*, volume 1, page 7, 2007.

[DFKE07a] Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian. Color image denoising via sparse 3d collaborative filtering with grouping constraint in luminance-chrominance space. In *2007 IEEE International Conference on Image Processing*, volume 1, pages I–313. IEEE, 2007.

[DFKE07b] Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian. Image denoising by sparse 3-d transform-domain collaborative filtering. *IEEE Transactions on image processing*, 2007.

[DJ94] David L. Donoho and Jain M. Johnstone. Ideal spatial adaptation by wavelet shrinkage. *Biometrika*, 81(3):425–455, 1994.

[DLBM14] Xuemei Ding, Yuhua Li, Ammar Belatreche, and Liam P Maguire. An experimental evaluation of novelty detection methods. *Neurocomputing*, 135:313–327, 2014.

[DMFML16] J. Matías Di Martino, Gabriele Facciolo, and Enric Meinhardt-Llopis. Poisson Image Editing. *Image Processing On Line*, 6:300–325, 2016.

[DMM07] Agnes Desolneux, Lionel Moisan, and Jean-Michel Morel. *From gestalt theory to image analysis: a probabilistic approach*. Springer Science & Business Media, 2007.

[DVF+09] Aram Danielyan, Markku Vehvilainen, Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian. Cross-color bm3d filtering of noisy raw data. In *2009 international workshop on local and non-local approximation in image processing*, pages 125–129. IEEE, 2009.

[DZ11] Bo Du and Liangpei Zhang. Random-selection-based anomaly detector for hyperspectral imagery. *IEEE Transactions on Geoscience and Remote sensing*, 49(5):1578–1589, 2011.

[EA06] Michael Elad and Michal Aharon. Image denoising via sparse and redundant representations over learned dictionaries. *IEEE Transactions on Image Processing*, 15(12):3736–3745, 2006.

[EA18] Thibaud Ehret and Pablo Arias. On the convergence of patchmatch and its variants. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1121–1129, 2018.

[EA20] Thibaud Ehret and Pablo Arias. Implementation of the vbm3d video denoising method and some variants. *Arxiv*, 2020.

[EAM17] Thibaud Ehret, Pablo Arias, and Jean-Michel Morel. Global patch search boosts video denoising. In *International Conference on Computer Vision Theory and Applications*, 2017.

[EDAF19] Thibaud Ehret, Axel Davy, Pablo Arias, and Gabriele Facciolo. Joint demosaicking and denoising by fine-tuning of bursts of raw images. In *IEEE ICCV*, 2019.

[EDM+19] Thibaud Ehret, Axel Davy, Jean-Michel Morel, Gabriele Facciolo, and Pablo Arias. Model-blind video denoising via frame-to-frame training. In *IEEE CVPR*, pages 11369–11378, 2019.

[EDMD19] Thibaud Ehret, Axel Davy, Jean-Michel Morel, and Mauricio Delbracio. Image anomalies: A review and synthesis of detection methods. *Journal of Mathematical Imaging and Vision*, 2019.

[Ehr18] T. Ehret. Automatic Detection of Internal Copy-Move Forgeries in Images. *IPOL*, 8:167–191, 2018.

[EL99] Alexei A Efros and Thomas K Leung. Texture synthesis by non-parametric sampling. In *ICCV*, 1999.

[EMA18] Thibaud Ehret, Jean-Michel Morel, and Pablo Arias. Non-local kalman: A recursive video denoising algorithm. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 3204–3208. IEEE, 2018.

[ESV12] Ehsan Elhamifar, Guillermo Sapiro, and Rene Vidal. See all by looking at a few: Sparse modeling for finding representative objects. In *CVPR*, 2012.

[Far09] H. Farid. Image forgery detection. *IEEE Signal processing magazine*, 26(2):16–25, 2009.

[FBDRP12] P. Ferrara, T. Bianchi, A. De Rosa, and A. Piva. Image forgery localization via fine-grained analysis of cfa artifacts. *IEEE TIFS*, 7(5):1566–1577, 2012.

[FBK15] Denis Fortun, Patrick Bouthemy, and Charles Kervrann. Sparse aggregation framework for optical flow estimation. In *Proceedings of the 5th Int. Conf. on Scale Space and Variational Methods in Computer Vision (SSVM)*, pages 323–334. Springer, 2015.

[FK18] Iuri Frosio and Jan Kautz. Statistical nearest neighbors for image denoising. *IEEE TIP*, 28(2):723–738, 2018.

[FPM17a] G. Facciolo, N. Pierazzo, and J. Morel. Conservative scale recomposition for multiscale denoising (the devil is in the high frequency detail). *SIAM Journal on Imaging Sciences*, 10(3):1603–1626, 2017.

[FPM17b] Gabriele Facciolo, Nicola Pierazzo, and Jean-Michel Morel. Conservative scale recomposition for multiscale denoising (the devil is in the high frequency detail). *SIAM Journal on Imaging Sciences*, 10(3):1603–1626, 2017.

[Fra99] Rich Franzen. Kodak lossless true color image suite. *source:* `http://r0k.us/graphics/kodak`, 4, 1999.

[GB10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[GC04] Arnon Goldman and Israel Cohen. Anomaly detection based on an iterative local statistics approach. *Signal Processing*, 84(7):1225–1229, 2004.

[GCPD16] Michaël Gharbi, Gaurav Chaurasia, Sylvain Paris, and Frédo Durand. Deep joint demosaicking and denoising. *ACM Transactions on Graphics (TOG)*, 35(6):191, 2016.

[Get11] Pascal Getreuer. Color demosaicing with contour stencils. In *2011 17th International Conference on Digital Signal Processing (DSP)*, pages 1–6. IEEE, 2011.

[Get12] Pascal Getreuer. Image Demosaicking with Contour Stencils. *Image Processing On Line*, 2:22–34, 2012.

[GGM11a] Bruno Galerne, Yann Gousseau, and Jean-Michel Morel. Micro-Texture Synthesis by Phase Randomization. *Image Processing On Line*, 1:213–237, 2011.

[GGM11b] Bruno Galerne, Yann Gousseau, and Jean-Michel Morel. Random phase textures: Theory and synthesis. *IEEE Transactions on image processing*, 20(1):257–267, 2011.

[GIM+99] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *VLDB*, volume 99, pages 518–529, 1999.

[GLA+10] Bart Goossens, Hiêp Luong, Jan Aelterman, Aleksandra Pižurica, and Wilfried Philips. A gpu-accelerated real-time nlmeans algorithm for denoising color video sequences. In *ACIVS*, pages 46–57. Springer, 2010.

[GM09] Bénédicte Grosjean and Lionel Moisan. A-contrario detectability of spots in textured backgrounds. *Journal of Mathematical Imaging and Vision*, 33(3):313–337, 2009.

[GMU18a] Clément Godard, Kevin Matzen, and Matt Uyttendaele. Deep burst denoising. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 538–554, 2018.

[GMU18b] Clément Godard, Kevin Matzen, and Matt Uyttendaele. Deep burst denoising. In *European Conference on Computer Vision*, pages 538–554, 2018.

[GMV08] Dashan Gao, Vijay Mahadevan, and Nuno Vasconcelos. The discriminant center-surround hypothesis for bottom-up saliency. In *Advances in neural information processing systems*, pages 497–504, 2008.

[GO07] Guy Gilboa and Stanley Osher. Nonlocal linear image regularization and supervised segmentation. *Multiscale Modeling & Simulation*, 6(2):595–630, 2007.

[GP15] Rafael Grompone and Viorica Pătrăucean. A contrario patch matching, with an application to keypoint matches validation. In *ICIP*. IEEE, 2015.

[GPMA18] Mario Gonzalez, Javier Preciozzi, Pablo Muse, and Andres Almansa. Joint denoising and decompression using cnn regularization. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018.

[Gul03] O. G. Guleryuz. Weighted overcomplete denoising. In *The Thrity-Seventh Asilomar Conference on Signals, Systems Computers, 2003*, volume 2, pages 1992–1996 Vol.2, Nov 2003.

[GYZ+18] Shi Guo, Zifei Yan, Kai Zhang, Wangmeng Zuo, and Lei Zhang. Toward convolutional blind denoising of real photographs. *arXiv preprint arXiv:1807.04686*, 2018.

[GZMT12] Stas Goferman, Lihi Zelnik-Manor, and Ayellet Tal. Context-aware saliency detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(10):1915–1926, 2012.

[GZZF14] Shuhang Gu, Lei Zhang, Wangmeng Zuo, and Xiangchu Feng. Weighted nuclear norm minimization with application to image denoising. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2862–2869, 2014.

[HBK+14] Michael Hornáček, Frederic Besse, Jan Kautz, Andrew Fitzgibbon, and Carsten Rother. Highly overparameterized optical flow using PatchMatch Belief Propagation. In *Proceedings of the 13th Europ. Conf. on Computer Vision (ECCV)*, pages 220–234. Springer, 2014.

[HC07] Y.-F. Hsu and S.-F. Chang. Image splicing detection using camera response function consistency and automatic segmentation. In *Int. conf. on Multimedia and Expo*, pages 28–31. IEEE, 2007.

[HEK+14] Felix Heide, Karen Egiazarian, Jan Kautz, Kari Pulli, Markus Steinberger, Yun-Ta Tsai, Mushfiqur Rouf, Dawid Pająk, Dikpal Reddy, Orazio Gallo, Jing Liu, and Wolfgang Heidrich. FlexISP. *ACM Transactions on Graphics*, 33(6):1–13, 11 2014.

[HHWB02] Simon Hawkins, Hongxing He, Graham Williams, and Rohan Baxter. Outlier detection using replicator neural networks. In *DaWaK*, 2002.

[HK18] David Honzátko and Martin Kruliš. Cuda implementation of bm3d. https://github.com/DawyD/bm3d-gpu, 2018.

[HK19] David Honzátko and Martin Kruliš. Accelerating block-matching and 3d filtering method for image denoising on gpus. *Journal of Real-Time Image Processing*, 16(6):2273–2287, 2019.

[HL08] Mark J Huiskes and Michael S Lew. The mir flickr retrieval evaluation. In *Proceedings of the 1st ACM international conference on Multimedia information retrieval*, pages 39–43. ACM, 2008.

[HLOE18] Minyoung Huh, Andrew Liu, Andrew Owens, and Alexei A Efros. Fighting fake news: Image splice detection via learned self-consistency. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 101–117, 2018.

[HN01] Toshifumi Honda and Shree K Nayar. Finding "anomalies" in an arbitrary image. In *ICCV*, 2001.

[HP06] Keigo Hirakawa and Thomas W Parks. Joint demosaicing and denoising. *IEEE Transactions on Image Processing*, 15(8):2146–2157, 2006.

[HRR+11] Kaiming He, Christoph Rhemann, Carsten Rother, Xiaoou Tang, and Jian Sun. A global sampling method for alpha matting. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 2049–2056. IEEE, 2011.

[HS12] Kaiming He and Jian Sun. Computing Nearest-Neighbor Fields via Propagation-Assisted KD-trees. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 111–118. IEEE, 2012.

[HSBZ15] Xun Huang, Chengyao Shen, Xavier Boix, and Qi Zhao. Salicon: Reducing the semantic gap in saliency prediction by adapting deep neural networks. In *ICCV*, 2015.

[HSG+16] Samuel W. Hasinoff, Dillon Sharlet, Ryan Geiss, Andrew Adams, Jonathan T. Barron, Florian Kainz, Jiawen Chen, and Marc Levoy. Burst photography for high dynamic range and low-light imaging on mobile cameras. *ACM Transactions on Graphics*, 35(6):1–12, nov 2016.

[HWW15] Yan Huang, Wei Wang, and Liang Wang. Bidirectional recurrent convolutional networks for multi-frame super-resolution. In *NIPS*. 2015.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 770–778, 2016.

[IK00] Laurent Itti and Christof Koch. A saliency-based search mechanism for overt and covert shifts of visual attention. *Vision research*, 40(10):1489–1506, 2000.

[IKN98] Laurent Itti, Christof Koch, and Ernst Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(11):1254–1259, 1998.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613. ACM, 1998.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456. PMLR, 07–09 Jul 2015.

[JGKL17] Qiyu Jin, Ion Grama, Charles Kervrann, and Quansheng Liu. Nonlocal means and optimal weights for noise removal. *SIAM SIIMS*, 10(4):1878–1920, 2017.

[JS09] Viren Jain and Sebastian Seung. Natural image denoising with convolutional networks. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 769–776. Curran Associates, Inc., 2009.

[Jun15] Stephen Junkins. *The compute architecture of intel® processor graphics gen9*, 2015.

[JWY+13] Huaizu Jiang, Jingdong Wang, Zejian Yuan, Yang Wu, Nanning Zheng, and Shipeng Li. Salient object detection: A discriminative regional feature integration approach. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2083–2090, 2013.

[KA11] Simon Korman and Shai Avidan. Coherency sensitive hashing. In *Proceedings of the IEEE Int. Conf. on Computer Vision (ICCV)*, pages 1607–1614. IEEE, 2011.

[Kal60] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 1960.

[Kay93] SM Kay. Fundamentals of statistical processing, volume i: Estimation theory: Estimation theory v. 1, 1993.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KBC07] Charles Kervrann, Jérôme Boulanger, and Pierrick Coupé. Bayesian non-local means filter, image redundancy and adaptive dictionaries for noise removal. In *International Conference on Scale Space and Variational Methods in Computer Vision*, pages 520–532. Springer, 2007.

[KCBI19] Chiman Kwan, Bryan Chou, and James F Bell III. Comparison of deep learning and conventional demosaicing algorithms for mastcam images. *Electronics*, 8(3):308, 2019.

[KH11] M. Klaudiny and A. Hilton. Cooperative patch-based 3D surface tracking. In *Proceedings of the Conf. for Visual Media Production (CVMP)*, pages 67–76, 2011.

[KHKP16] Teresa Klatzer, Kerstin Hammernik, Patrick Knobelreiter, and Thomas Pock. Learning joint demosaicing and denoising based on sequential energy minimization. In *2016 IEEE International Conference on Computational Photography (ICCP)*, pages 1–11. IEEE, 2016.

[KKHP17] Erich Kobler, Teresa Klatzer, Kerstin Hammernik, and Thomas Pock. Variational networks: Connecting variational methods and deep learning. In Volker Roth and Thomas Vetter, editors, *Pattern Recognition*, pages 281–293, Cham, 2017. Springer International Publishing.

[KL18a] Filippos Kokkinos and Stamatios Lefkimmiatis. Deep image demosaicking using a cascade of convolutional residual denoising networks. In *The European Conference on Computer Vision (ECCV)*, September 2018.

[KL18b] Filippos Kokkinos and Stamatios Lefkimmiatis. Iterative residual network for deep joint image demosaicking and denoising. *CoRR*, abs/1807.06403, 2018.

[KMTO13] Daisuke Kiku, Yusuke Monno, Masayuki Tanaka, and Masatoshi Okutomi. Residual interpolation for color image demosaicking. In *2013 IEEE International Conference on Image Processing*, pages 2304–2308. IEEE, 2013.

[KMTO14] Daisuke Kiku, Yusuke Monno, Masayuki Tanaka, and Masatoshi Okutomi. Minimized-laplacian residual interpolation for color image demosaicking. In *Digital Photography X*, volume 9023, page 90230L. International Society for Optics and Photonics, 2014.

[KNJF14] Daniel Khashabi, Sebastian Nowozin, Jeremy Jancsary, and Andrew W. Fitzgibbon. Joint Demosaicing and Denoising via Learned Nonparametric Random Fields. *IEEE Transactions on Image Processing*, 23(12):4968–4981, 12 2014.

[Kum03] Ajay Kumar. Neural network based detection of local textile defects. *Pattern Recognition*, 36(7):1645–1659, 2003.

[KZN08] Neeraj Kumar, Li Zhang, and Shree Nayar. What is a good nearest neighbors algorithm for finding similar patches in images? In *Proceedings of the Europ. Conf. on Computer Vision*, pages 364–378. Springer, 2008.

[LBM13a] Marc Lebrun, Antoni Buades, and Jean-Michel Morel. A Nonlocal Bayesian Image Denoising Algorithm. *SIAM Journal on Imaging Sciences*, 6(3):1665–1688, 2013.

[LBM13b] Marc Lebrun, Antoni Buades, and Jean-Michel Morel. Implementation of the "Non-Local Bayes" (NL-Bayes) Image Denoising Algorithm. *Image Processing On Line*, 3:1–42, 2013.

[LBM13c] Marc Lebrun, Antoni Buades, and Jean-Michel Morel. A nonlocal bayesian image denoising algorithm. *SIAM Journal on Imaging Sciences*, 6(3):1665–1688, 2013.

[LCBM12] Marc Lebrun, Miguel Colom, Antoni Buades, and Jean-Michel Morel. Secrets of image denoising cuisine. *Acta Numerica*, 21(1):475–576, 2012.

[LCM15] Marc Lebrun, Miguel Colom, and Jean-Michel Morel. The noise clinic: a blind image denoising algorithm. *Image Processing On Line*, 5:1–54, 2015.

[Leb12] Marc Lebrun. An Analysis and Implementation of the BM3D Image Denoising Method. *Image Processing On Line*, 2:175–213, 2012.

[Lef17] S. Lefkimmiatis. Non-local color image denoising with convolutional neural networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5882–5891, July 2017.

[LF10] Ce Liu and William T Freeman. A high-quality video denoising algorithm based on reliable motion estimation. In *Proceedings of the Europ. Conf. on Computer Vision*, pages 706–719. Springer, 2010.

[LGvGRM14] José Lezama, Rafael Grompone von Gioi, Gregory Randall, and Jean-Michel Morel. Finding vanishing points via point alignments in image primal and dual domains. In *CVPR*, 2014.

[LHTT09] Z. Lin, J. He, X. Tang, and C.-K. Tang. Fast, automatic and fine-grained tampered jpeg image detection via dct coefficient analysis. *Pattern Recognition*, 42(11):2492–2501, 2009.

[LJH+19] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *CoRR*, abs/1908.03265, 2019.

[LMH+18] Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, and Timo Aila. Noise2noise: Learning image restoration without clean data. In *International Conference on Machine Learning*, pages 2971–2980, 2018.

[Low85] David Lowe. *Perceptual organization and visual recognition*. Kluwer Academic Publishers, 1985.

[Low99] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. IEEE, 1999.

[Low04] David G Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[LSK+17] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. In *IEEE Conference on computer vision and pattern recognition workshops*, pages 136–144, 2017.

[LWF+18] Ding Liu, Bihan Wen, Yuchen Fan, Chen Change Loy, and Thomas S Huang. Non-local recurrent network for image restoration. In *NIPS*, 2018.

[LWTS07] G. Li, Q. Wu, D. Tu, and S. Sun. A sorted neighborhood approach for detecting duplicated regions in image forgeries based on dwt and svd. In *Int. conf. on Multimedia and Expo*, pages 1750–1753. IEEE, 2007.

[LY10] W. Li and N. Yu. Rotation robust detection of copy-move forgery. In *ICIP*, 2010.

[LYMD13] J. Lu, H. Yang, D. Min, and M. N. Do. Patch match filter: Efficient edge-aware filtering meets randomized search for fast correspondence field estimation. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1854–1861, 2013.

[LYT+14] Ziwei Liu, Lu Yuan, Xiaoou Tang, Matt Uyttendaele, and Jian Sun. Fast burst images denoising. *ACM Transactions on Graphics (TOG)*, 2014.

[LYT+17] Z. Liu, R. A. Yeh, X. Tang, Y. Liu, and A. Agarwala. Video frame synthesis using deep voxel flow. In *IEEE ICCV*, 2017.

[LZD11] Wen Li, Jun Zhang, and Qiong-Hai Dai. Video denoising using shape-adaptive sparse representation over similar spatio-temporal patches. *Signal Processing: Image Communication*, 26:250–265, April 2011.

[MBC+18a] Ben Mildenhall, Jonathan T. Barron, Jiawen Chen, Dillon Sharlet, Ren Ng, and Rob Carroll. Burst denoising with kernel prediction networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[MBC+18b] Ben Mildenhall, Jonathan T Barron, Jiawen Chen, Dillon Sharlet, Ren Ng, and Robert Carroll. Burst denoising with kernel prediction networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2502–2510, 2018.

[MBFE11] Matteo Maggioni, Giacomo Boracchi, Alessandro Foi, and Karen Egiazarian. Video Denoising Using Separable 4D Nonlocal Spatiotemporal Transforms. In *Proc. of SPIE*, number 7870, 2011.

[MBFE12] Matteo Maggioni, Giacomo Boracchi, Alessandro Foi, and Karen Egiazarian. Video denoising, deblocking, and enhancement through separable 4-D nonlocal spatiotemporal transforms. *Image Processing, IEEE Transactions on*, 21(9):3952–3966, 2012.

[MBP+09] J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman. Non-local sparse models for image restoration. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2272–2279, Sept 2009.

[MC13] Gal Mishne and Israel Cohen. Multiscale anomaly detection using diffusion maps. *IEEE Journal of selected topics in signal processing*, 7(1):111–123, 2013.

[MC14] Gal Mishne and Israel Cohen. Multiscale anomaly detection using diffusion maps and saliency score. In *ICASSP*, 2014.

[MDW+17] Kede Ma, Zhengfang Duanmu, Qingbo Wu, Zhou Wang, Hongwei Yong, Hongliang Li, and Lei Zhang. Waterloo Exploration Database: New challenges for image quality assessment models. *IEEE Transactions on Image Processing*, 26(2):1004–1016, Feb. 2017.

[MF11a] Markku Makitalo and Alessandro Foi. A closed-form approximation of the exact unbiased inverse of the anscombe variance-stabilizing transformation. *Transactions on Image Processing*, 20(9):2697–2698, 2011.

[MF11b] Markku Makitalo and Alessandro Foi. Optimal inversion of the anscombe transformation in low-count poisson image denoising. *Transactions on Image Processing*, 20(1):99–109, 2011.

[MFTM01] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. 8th Int'l Conf. Computer Vision*, volume 2, pages 416–423, July 2001.

[MKEF13] Matteo Maggioni, Vladimir Katkovnok, Karen Egiazarian, and Alessandro Foi. A nonlocal transform-domain filter for volumetric data denoising and reconstruction. *IEEE Transactions on Image Processing*, 22(1):119–133, 2013.

[MKRH11] Rafat Mantiuk, Kil Joong Kim, Allan G Rempel, and Wolfgang Heidrich. Hdr-vdp-2: a calibrated visual metric for visibility and quality predictions in all luminance conditions. In *ACM Transactions on graphics (TOG)*, volume 30, page 40. ACM, 2011.

[MKTO15] Yusuke Monno, Daisuke Kiku, Masayuki Tanaka, and Masatoshi Okutomi. Adaptive residual interpolation for color image demosaicking. In *2015 IEEE International Conference on Image Processing (ICIP)*, pages 3861–3865. IEEE, 2015.

[ML99] Pierre Moulin and Juan Liu. Analysis of multiresolution image denoising schemes using generalized gaussian and complexity priors. *IEEE Transactions on Information Theory*, 45(3):909–919, 1999.

[ML09] Marius Muja and David G Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)*, 2(331-340):2, 2009.

[MP13] Adrián Márques and Alvaro Pardo. Implementation of non local means filter in gpus. In *Iberoamerican Congress on Pattern Recognition*, pages 407–414. Springer, 2013.

[MS03] Markos Markou and Sameer Singh. Novelty detection: a review –part 1: statistical approaches. *Signal processing*, 83(12):2481–2497, 2003.

[MS04] Lionel Moisan and Bérenger Stival. A probabilistic criterion to detect rigid point matches between two images and estimate the fundamental matrix. *International Journal of Computer Vision*, 57(3):201–218, 2004.

[MS05] Mona Mahmoudi and Guillermo Sapiro. Fast image and video denoising via nonlocal means of similar neighborhoods. *IEEE SPL*, 12(12):839–842, 2005.

[MSH08] Morten Mørup, Mikkel N Schmidt, and Lars K Hansen. Shift invariant sparse coding of image and music data. *Submitted to Journal of Machine Learning Research*, 2008.

[MSMF14] Matteo Maggioni, Enrique Sánchez-Monge, and Alessandro Foi. Joint removal of random and fixed-pattern noise through spatiotemporal video filtering. *IEEE Transactions on Image Processing*, 23(10):4282–4296, 2014.

[MSS+14] Kiran Murthy, Michael Shearn, Byron D Smiley, Alexandra H Chau, Josh Levine, and M Dirk Robinson. Skysat-1: very high-resolution imagery from a small satellite. In *Sensors, Systems, and Next-Generation Satellites*, volume 9241. SPIE, 2014.

[MSY16] Xiaojiao Mao, Chunhua Shen, and Yu-Bin Yang. Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2802–2810. Curran Associates, Inc., 2016.

[MTZM13] Ran Margolin, Ayellet Tal, and Lihi Zelnik-Manor. What makes a patch distinct? In *CVPR*, 2013.

[MVOP11] Naila Murray, Maria Vanrell, Xavier Otazu, and C Alejandro Parraga. Saliency estimation using a non-parametric low-level vision model. In *CVPR*, 2011.

[MZI13] Inbar Mosseri, Maria Zontak, and Michal Irani. Combining the power of internal and external denoising. In *IEEE International Conference on Computational Photography*, pages 1–9, 2013.

[NCS04] T.-T. Ng, S.-F. Chang, and Q. Sun. Blind detection of photomontage using higher order statistics. In *ISCAS*, volume 5. IEEE, 2004.

[NGvGCM18] T. Nikoukhah, R. Grompone von Gioi, M. Colom, and J.-M. Morel. Automatic jpeg grid detection with controlled false alarms, and its image forensic applications. In *IEEE MIPR*, pages 378–383. IEEE, 2018.

[NVI09] NVIDIA. *NVIDIA OpenCL Best Practices Guide*, 2009.

[OA12] Igor Olonetsky and Shai Avidan. TreeCANN - kd-tree Coherence Approximate Nearest Neighbor algorithm. In *Proceedings of the Europ. Conf. on Computer Vision (ECCV)*, pages 602–615. Springer, 2012.

[OD+13] Stephen O'Hara, Bruce Draper, et al. Are you using the right approximate nearest neighbor algorithm? In *Applications of Computer Vision (WACV), 2013 IEEE Workshop on*, pages 9–14. IEEE, 2013.

[PCC10] Der-Baau Perng, Ssu-Han Chen, and Yuan-Shuo Chang. A novel internal thread defect auto-inspection system. *International Journal of Advanced Manufacturing Technology*, 47(5-8):731–743, 2010.

[PCCT14] Marco AF Pimentel, David A Clifton, Lei Clifton, and Lionel Tarassenko. A review of novelty detection. *Signal Processing*, 99:215–249, 2014.

[PE09] Matan Protter and Michael Elad. Image sequence denoising via sparse and redundant representations. *IEEE Transactions on Image Processing*, 18(1):27–35, 2009.

[PE16] Vardan Papyan and Michael Elad. Multi-scale patch-based image restoration. *IEEE Transactions on image processing*, 25(1):249–261, 2016.

[PF05] A. Popescu and H. Farid. Exposing digital forgeries in color filter array interpolated images. *Trans. on Signal Processing*, 53(10):3948–3959, 2005.

[PGB03] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. *ACM Transactions on graphics (TOG)*, 22(3):313–318, 2003.

[PGvG12] V Patraucean, P Gurdjos, and R Grompone von Gioi. A parameterless ellipse and line segment detector with enhanced ellipse fitting. In *ECCV*, 2012.

[PGvGO13] Viorica Patraucean, Rafael Grompone von Gioi, and Maks Ovsjanikov. Detection of mirror-symmetric image patches. In *CVPR*, 2013.

[PKL+09] Sung Hee Park, Hyung Suk Kim, Steven Lansel, Manu Parmar, and Brian A Wandell. A case for denoising before demosaicking color filter array data. In *2009 Conference Record of the Forty-Third Asilomar Conference on Signals, Systems and Computers*, pages 860–864. IEEE, 2009.

[PL10] Xunyu Pan and Siwei Lyu. Region duplication detection using image feature matching. *IEEE Transactions on Information Forensics and Security*, 5(4):857–867, 2010.

[PMF17] Nicola Pierazzo, Jean-Michel Morel, and Gabriele Facciolo. Multi-Scale DCT Denoising. *Image Processing On Line*, 2017.

[PPR+17] Sergio G. Pfleger, Patricia D. M. Plentz, Rodrigo C. O. Rocha, Alyson D. Pereira, and Márcio Castro. Real-time video denoising on multicores and gpus with kalman-based and bilateral filters fusion. *Journal of Real-Time Image Processing*, Feb 2017.

[PR17] Tobias Plotz and Stefan Roth. Benchmarking denoising algorithms with real photographs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1586–1595, 2017.

[PR18] Tobias Plötz and Stefan Roth. Neural nearest neighbors networks. In *NIPS*, 2018.

[PSWS03] J. Portilla, V. Strela, M.J. Wainwright, and E.P. Simoncelli. Image denoising using scale mixtures of gaussians in the wavelet domain. *Image Processing, IEEE Transactions on*, 12(11):1338–1351, Nov 2003.

[PTPC+17] Jordi Pont-Tuset, Federico Perazzi, Sergi Caelles, Pablo Arbeláez, Alexander Sorkine-Hornung, and Luc Van Gool. The 2017 davis challenge on video object segmentation. *arXiv:1704.00675*, 2017.

[QDF+17] Peng Qiao, Yong Dou, Wensen Feng, Rongchun Li, and Yunjin Chen. Learning non-local image diffusion for image denoising. In *Proceedings of the 25th ACM International Conference on Multimedia*, MM '17, pages 1847–1855, New York, NY, USA, 2017. ACM.

[RB17] Anurag Ranjan and Michael J Black. Optical flow estimation using a spatial pyramid network. In *IEEE CVPR*, 2017.

[RDDM18] Lara Raad, Axel Davy, Agnès Desolneux, and Jean-Michel Morel. A survey of exemplar-based texture synthesis. *Annals of Mathematical Sciences and Applications*, 3(1):89–148, 2018.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[RGvG18] M. Rodríguez and R. Grompone von Gioi. Affine invariant image comparison under repetitive structures. In *IEEE ICIP*. IEEE, 2018.

[RMD+13] Nicolas Riche, Matei Mancas, Matthieu Duvinage, Makiese Mibulumukini, Bernard Gosselin, and Thierry Dutoit. RARE2012: A multi-scale rarity-based saliency detection with its comparative statistical analysis. *Signal Processing: Image Communication*, 28(6):642–658, 2013.

[ROD14] I. Rey Otero and M. Delbracio. Anatomy of the SIFT Method. *IPOL*, 4:370–396, 2014.

[ROF92a] Leonid I. Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Phys. D*, 60(1-4):259–268, November 1992.

[ROF92b] Leonid I Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: nonlinear phenomena*, 60(1-4):259–268, 1992.

[Rot05] Roth, Stefan and Black, Michael J. Fields of experts: a framework for learning image priors. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 860–867 vol. 2, June 2005.

[SB11] BL Shivakumar and LDSS Baboo. Detection of region duplication forgery in digital images using surf. *IJCSI International Journal of Computer Science Issues*, 8(4), 2011.

[SCC18] Nai-Sheng Syu, Yu-Sheng Chen, and Yung-Yu Chuang. Learning deep convolutional networks for demosaicing. *arXiv preprint arXiv:1802.03769*, 2018.

[Sch92] Dale Schumacher. General filtered image rescaling. In DAVID KIRK, editor, *Graphics Gems III (IBM Version)*, pages 8 – 16. Morgan Kaufmann, San Francisco, 1992.

[SCI18] Assaf Shocher, Nadav Cohen, and Michal Irani. Zero-Shot Super-Resolution Using Deep Internal Learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[SDA14] C. Sutour, C.-A. Deledalle, and J.-F. Aujol. Adaptive regularization of the nl-means: Application to image and video denoising. *IEEE Transactions on Image Processing*, 23(8):3506–3521, Aug 2014.

[SDW+17] Shuochen Su, Mauricio Delbracio, Jue Wang, Guillermo Sapiro, Wolfgang Heidrich, and Oliver Wang. Deep video deblurring for hand-held cameras. In *IEEE CVPR*, 2017.

[SM09] Hae Jong Seo and Peyman Milanfar. Static and space-time visual saliency detection by self-resemblance. *Journal of vision*, 9(12):15–15, 2009.

[SMD17] Venkataraman Santhanam, Vlad I Morariu, and Larry S Davis. Generalized deep image to image regression. In *IEEE CVPR*, 2017.

[SPMLF13] Javier Sánchez Pérez, Enric Meinhardt-Llopis, and Gabriele Facciolo. TV-L1 Optical Flow Estimation. *Image Processing On Line*, 2013.

[SR14] U. Schmidt and S. Roth. Shrinkage fields for effective image restoration. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2774–2781, June 2014.

[SSW+17] Thomas Schlegl, Philipp Seeböck, Sebastian M. Waldstein, Ursula Schmidt-Erfurth, and Georg Langs. Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery. In *IPMI*, 2017.

[ST11] J. Sun and M. F. Tappen. Learning non-local range markov random field for image restoration. In *IEEE CVPR*, 2011.

[SVB18] Mehdi S. M. Sajjadi, Raviteja Vemulapalli, and Matthew Brown. Frame-Recurrent Video Super-Resolution. In *IEEE CVPR*, 2018.

[SZ15] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[TD98] David MJ Tax and Robert PW Duin. Outlier detection using classifier instability. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, 1998.

[TD20] Zachary Teed and Jia Deng. RAFT: Recurrent all-pairs field transforms for optical flow. In *European Conference on Computer Vision*, 2020.

[TDV19a] Matias Tassano, Julie Delon, and Thomas Veit. Dvdnet: A fast network for deep video denoising. In *IEEE ICIP*, 2019.

[TDV19b] Matias Tassano, Julie Delon, and Thomas Veit. Fastdvdnet: Towards real-time video denoising without explicit motion estimation. volume abs/1907.01361v1, 2019.

[TH99] D-M Tsai and C-Y Hsieh. Automated surface inspection for directional textures. *Image and vision computing*, 18(1):49–62, 1999.

[TH03] Du-Ming Tsai and Tse-Yun Huang. Automated surface inspection for statistical textures. *Image and Vision computing*, 21(4):307–323, 2003.

[THCB95] Lionel Tarassenko, Paul Hayton, Nicholas Cerneaz, and Michael Brady. Novelty detection for the identification of masses in mammograms. In *International Conference on Artificial Neural Networks*, 1995.

[TM98] Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In *Computer Vision, 1998. Sixth International Conference on*, pages 839–846. IEEE, 1998.

[TRH11] Hamed Rezazadegan Tavakoli, Esa Rahtu, and Janne Heikkilä. Fast and efficient saliency detection using sparse sampling and kernel density estimation. In *Scandinavian Conference on Image Analysis*, 2011.

[TRJ+19] Alessio Tonioni, Oscar Rahnama, Thomas Joy, Luigi Di Stefano, Thalaiyasingam Ajanthan, and Philip Torr. Learning to Adapt for Stereo. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[TRU98] Philippe Thevenaz, Urs E Ruttimann, and Michael Unser. A pyramid approach to subpixel registration based on intensity. *IEEE transactions on image processing*, 7(1):27–41, 1998.

[TTP+19] Alessio Tonioni, Fabio Tosi, Matteo Poggi, Stefano Mattoccia, and Luigi Di Stefano. Real-time self-adaptive deep stereo. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[TYLX17] Ying Tai, Jian Yang, Xiaoming Liu, and Chunyan Xu. Memnet: A persistent memory network for image restoration. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 4549–4557, 2017.

[TZZZ17] Runjie Tan, Kai Zhang, Wangmeng Zuo, and Lei Zhang. Color image demosaicking via deep residual learning. In *IEEE Int. Conf. Multimedia and Expo (ICME)*, 2017.

[UVL18] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Deep image prior. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9446–9454, 2018.

[VF08] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms. http://www.vlfeat.org/, 2008.

[VFM19] Diego Valsesia, Giulia Fracastoro, and Enrico Magli. Image denoising with graph-convolutional neural networks. In *IEEE International Conference on Image Processing*, pages 2399–2403. IEEE, 2019.

[VGJMR10] Rafael Grompone Von Gioi, Jeremie Jakubowicz, Jean-Michel Morel, and Gregory Randall. Lsd: A fast line segment detector with a false detection control. *IEEE Trans. PAMI*, 2010.

[VTL16] R. Vemulapalli, O. Tuzel, and M. Liu. Deep gaussian conditional random field network: A model-based deep network for discriminative denoising. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4801–4809, June 2016.

[WGDE+19] Bartlomiej Wronski, Ignacio Garcia-Dorado, Manfred Ernst, Damien Kelly, Michael Krainin, Chia-Kai Liang, Marc Levoy, and Peyman Milanfar. Handheld multi-frame super-resolution. *ACM Transactions on Graphics*, 38(4):1–18, jul 2019.

[WGY+06] Jin Wang, Yanwen Guo, Yiting Ying, Yanli Liu, and Qunsheng Peng. Fast non-local algorithm for image denoising. In *IEEE ICIP*, pages 1429–1432, 2006.

[WL15] YQ Wang and N Limare. A fast c++ implementation of neural network backpropagation training algorithm: Application to bayesian optimal image demosaicking, image processing on line,(2015), 2015.

[WLPB17] Bihan Wen, Yanjun Li, Luke Pfister, and Yoram Bresler. Joint adaptive sparsity and low-rankness on the fly: an online tensor reconstruction scheme for video denoising. In *IEEE ICCV*, 2017.

[WR04] Alle Meije Wink and Jos BTM Roerdink. Denoising functional mr images: a comparison of wavelet denoising and gaussian smoothing. *IEEE transactions on medical imaging*, 23(3):374–387, 2004.

[WS17] Tianxiong Wang and Yushi Sun. Gpu-accelerated denoising with bm3d. https://github.com/JeffOwOSun/gpu-bm3d, 2017.

[WXW18] Xiaoyun Wang, Ke Xu, and Dong Wang. Accelerating block-matching and 3d filtering-based image denoising algorithm on fpgas. In *IEEE ICSP*, pages 235–240. IEEE, 2018.

[WZS+16] B. Wen, Y. Zhu, R. Subramanian, T.-T. Ng, X. Shen, and S. Winkler. Coverage—a novel database for copy-move forgery detection. In *ICIP*. IEEE, 2016.

[XCW+19] Tianfan Xue, Baian Chen, Jiajun Wu, Donglai Wei, and William T. Freeman. Video enhancement with task-oriented flow. *IJCV*, 127(8):1106–1125, Aug 2019.

[XM07] Xianghua Xie and Majid Mirmehdi. Texems: Texture exemplars for defect detection on random textured surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(8):1454–1464, 2007.

[XPL07] Yongqing Xin, Miroslaw Pawlak, and Simon Liao. Accurate computation of zernike moments in polar coordinates. *IEEE Transactions on Image Processing*, 16(2):581–587, 2007.

[XTWZ15] Lingxi Xie, Qi Tian, Jingdong Wang, and Bo Zhang. Image classification with max-sift descriptors. In *International conference on acoustics, speech and signal processing*, 2015.

[YHHZ20] Hongwei Yong, Jianqiang Huang, Xiansheng Hua, and Lei Zhang. Gradient centralization: A new optimization technique for deep neural networks. *CoRR*, abs/2004.01461, 2020.

[Yia93] Peter N Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *SODA*, volume 93, pages 311–321, 1993.

[YS18] D. Yang and J. Sun. Bm3d-net: A convolutional neural network for transform-domain collaborative filtering. *IEEE Signal Processing Letters*, 25(1):55–59, Jan 2018.

[YSM12] Guoshen Yu, G. Sapiro, and S. Mallat. Solving inverse problems with piecewise linear estimators: From gaussian mixture models to structured sparsity. *Transactions on Image Processing*, 21(5), 2012.

[ZC10] Maria Zontak and Israel Cohen. Defect detection in patterned wafers using anisotropic kernels. *Machine Vision and Applications*, 21(2):129–141, 2010.

[ZFSOM07] Bo Zhang, MJ Fadili, J-L Starck, and J-C Olivo-Marin. Multiscale variance-stabilizing transform for mixed-poisson-gaussian processes and its applications in bioimaging. In *IEE ICIP 2007*, 2007.

[ZGFK17] Hang Zhao, Orazio Gallo, Iuri Frosio, and Jan Kautz. Loss Functions for Image Restoration With Neural Networks. *IEEE Transactions on Computational Imaging*, 3(1):47–57, 3 2017.

[ZLBH19] Michael Zhang, James Lucas, Jimmy Ba, and Geoffrey E Hinton. Lookahead optimizer: k steps forward, 1 step back. In *Advances in Neural Information Processing Systems*, pages 9597–9608, 2019.

[ZLZ+17] Zhiyuan Zha, Xin Liu, Ziheng Zhou, Xiaohua Huang, Jingang Shi, Zhenhong Shang, Lan Tang, Yechao Bai, Qiong Wang, and Xinggan Zhang. Image denoising via group sparsity residual constraint. In *ICASSP*, 2017.

[ZN13] Wan-Lei Zhao and Chong-Wah Ngo. Flip-invariant sift for copy and object detection. *IEEE Transactions on Image Processing*, 22(3):980–991, 2013.

[ZPB07] Christopher Zach, Thomas Pock, and Horst Bischof. A duality based approach for realtime tv-l 1 optical flow. In *Joint Pattern Recognition Symposium*. Springer, 2007.

[ZSC19] Magauiya Zhussip, Shakarim Soltanayev, and Se Young Chun. Theoretical analysis on noise2noise using stein's unbiased risk estimator for gaussian denoising: Towards unsupervised training with clipped noisy images. *CoRR*, abs/1902.02452, 2019.

[ZSS+18] Amir R Zamir, Alexander Sax, William Shen, Leonidas J Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3712–3722, 2018.

[ZW11] D. Zoran and Y. Weiss. From learning models of natural image patches to whole image restoration. In *2011 International Conference on Computer Vision*, pages 479–486, Nov 2011.

[ZWBL11] Lei Zhang, Xiaolin Wu, Antoni Buades, and Xin Li. Color demosaicking by local directional interpolation and nonlocal adaptive thresholding. *Journal of Electronic Imaging*, 20(2):1 – 17, 2011.

[ZZC+17a] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising. *IEEE Transactions on Image Processing*, 26(7):3142–3155, 7 2017.

[ZZC+17b] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. *IEEE Transactions on Image Processing*, 26(7):3142–3155, 2017.

[ZZZ18] Kai Zhang, Wangmeng Zuo, and Lei Zhang. Ffdnet: Toward a fast and flexible solution for cnn-based image denoising. *Transactions on Image Processing*, 27(9):4608–4622, 2018.

université PARIS-SACLAY | École doctorale de mathématiques Hadamard (EDMH)

Titre : Débruitage vidéo et applications

Mots clés : Débruitage, Débruitage à patches, Apprentissage profond, Démosaïquage, Détection d'anomalies, PatchMatch, Détection de falsification

Résumé : Cette thèse est dédiée au débruitage vidéo. La première partie se concentre sur les méthodes à patches pour le débruitage de vidéo. Nous étudions en détail VBM3D, une méthode populaire de débruitage vidéo, pour comprendre les mécanismes qui ont fait son succès. Nous présentons aussi une implémentation temps-réel sur carte graphique de cette méthode. Nous étudions ensuite l'impacte de la recherche de patches pour le débruitage vidéo et en particulier comment une recherche globale peut améliorer la qualité du débruitage. Enfin, nous proposons une nouvelle méthode causale et récursive appelée NL-Kalman qui produit une très bonne consistance temporelle. Dans la deuxième partie, nous étudions les méthodes d'apprentissage pour le débruitage. Nous présentons l'une des toutes premières architecture de réseau qui est compétitive avec l'état de l'art. Nous montrons aussi que les méthodes basées sur l'apprentissage profond offrent de nouvelles opportunités. En particulier, il devient possible de débruiter sans connaître le modèle du bruit. Grâce à la méthode proposée, même les vidéos traitées par une chaîne de traitement inconnue peuvent être débruitées. Nous étudions aussi le cas de données mosaïquées. En particulier, nous montrons que les réseaux de neurones sont largement supérieurs aux méthodes précédentes. Nous proposons aussi une nouvelle méthode d'apprentissage pour le démosaïquage sans avoir besoin de vérité terrain. Dans une troisième partie nous présentons différentes applications aux techniques utilisées pour le débruitage. Le premier problème étudié est la détection d'anomalie. Nous montrons que ce problème peut être ramené à détecter des anomalies dans du bruit. Nous regardons aussi la détection de falsifications et en particulier la détection de copié-collé. Tout comme le débruitage à patches, ce problème peut être résolu à l'aide d'une recherche de patches similaires. Pour cela, nous étudions en détail PatchMatch et l'utilisons pour détecter des falsifications. Nous présentons aussi une méthode basée sur une association de patches parcimonieuse.

Title : Video denoising and applications

Keywords : Denoising, Patch-based denoising, Deep learning, Demosaicking, Anomaly detection, PatchMatch, Forgery detection

Abstract : This thesis studies the problem of video denoising. In the first part we focus on patch-based video denoising methods. We study in details VBM3D, a popular video denoising method, to understand the mechanisms that made its success. We also present a real-time implementation on GPU of this method. We then study the impact of patch search in video denoising and in particular how searching for similar patches in the entire video, a global patch search, improves the denoising quality. Finally, we propose a novel causal and recursive method called NL-Kalman that produces very good temporal consistency. In the second part, we look at the newer trend of deep learning for image and video denoising. We present one of the first neural network architecture, using temporal self-similarity, competitive with state-of-the-art patch-based video denoising methods. We also show that deep learning offers new opportunities. In particular, it allows for denoising without knowing the noise model. We propose a framework that allows denoising of videos that have been through an unknown processing pipeline. We then look at the case of mosaicked data. In particular, we show that deep learning is undeniably superior to previous approaches for demosaicking. We also propose a novel training process for demosaicking without ground-truth based on multiple raw acquisition. This allows training for real case applications. In the third part we present different applications taking advantage of mechanisms similar those studied for denoising. The first problem studied is anomaly detection. We show that this problem can be reduced to detecting anomalies in noise. We also look at forgery detection and in particular copy-paste forgeries. Just like for patch-based denoising, solving this problem requires searching for similar patches. For that, we do an in-depth study of PatchMatch and see how it can be used for detecting forgeries. We also present an efficient method based on sparse patch matching.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France