Object detection and traffic prediction using Deep Learning on compressed road images and videos

Benjamin Deguerre

Normandie Université

THESE

Pour obtenir le grade de Docteur de Normandie Université

Spécialité Informatique

Préparée au sein de l'INSA de Rouen Normandie, LITIS

Object detection and traffic prediction using Deep Learning on compressed road images and videos

Détection d'objets et prédiction du trafic routier à l'aide de l'apprentissage profond sur des images et des vidéos compressées de scènes routières.

Présentée et soutenue par

Benjamin DEGUERRE

Thèse soutenue publiquement le 25 Novembre 2021
devant le jury composé de

Mr Vincent FREMONT	Professeur, Centrale Nantes, LS2N	Rapporteur
Mr David PICARD	Habilité à diriger des recherches, Ecoles des Ponts ParisTech	Rapporteur
Mme Marianne CLAUSEL	Professeure, IECL	Examinatrice
Mr Franck DAVOINE	Habilité à diriger des recherches, CNRS	Examinateur
Mr Clément CHATELAIN	Habilité à diriger des recherches, INSA de Rouen Normandie, LITIS	Co-encadrant
Mr Gilles GASSO	Professeur, INSA de Rouen Normandie, LITIS	Directeur de thèse

Thèse dirigée par Gilles Gasso et Clément Chatelain, LITIS

INSA ROUEN NORMANDIE

MIIS

litis

Abstract

The PhD thesis is a CIFRE carried out with Actemium Paris Transport, a company that operates in the field of Intelligent Transport Systems (ITS) and, in particular, provides solutions for the surveillance of road tunnels. In the thesis, we address the learning of efficient deep learning models that directly process compressed images/videos to lower the computation resource requirements and to allow for large scale deployment of the solutions. More specifically, we target two types of compression, JPEG image compression and MPEG4 part-2 video compression, for two specific applications: object detection and traffic flow rate estimation.

The first contribution focuses on object detection in JPEG compressed images. As the JPEG algorithm compresses the images from a *spatial* representation into a tiled *frequency* space, the main challenge is to design detection models able to correctly estimate the *position* of objects based on the frequency representation. Using JPEG compressed images as inputs, we investigate deep learning architectures for object detection and demonstrate a $\times 1.7$ speed up at detection time, while only reducing the detection performance by 5.5%. Moreover, we empirically demonstrate that only part of the compressed information, namely the luminance component, is required to match the accuracy of the full input methods.

Our second contribution addresses the problem of estimating the flow rate (number of vehicles/unit of time) from MPEG4 part-2 compressed video streams issued from road surveillance cameras. The MPEG4 part-2 compression algorithm uses a coarse representation of the pixel flow across frames to reduce the size of the videos to be encoded. Therefore, the approximate flow representation appears relevant to estimate the flow rate, while reducing the computation and memory requirements. We propose multiple end-to-end deep learning architectures using this coarse pixel flow representation as input. Using these models, we demonstrate that predicting the flow rate directly from MPEG4 part-2 compressed video streams can be achieved, while reaching improved accuracy in comparison with a more classical RGB-based model. We also show an impressive speed up of $\times 3200$. Furthermore, as training data may be scarce due to practical constraints, we explore domain adaptation to transfer learned models from one camera to another and provide with a thorough analysis of the constraints that may impede such transfer.

Résumé

Cette thèse est une CIFRE réalisée avec Actemium Paris Transport, une société qui évolue dans le domaine des Systèmes de Transport Intelligents (STI) et, en particulier, fournit des solutions logicielles pour la surveillance des tunnels routiers. Dans cette thèse, nous nous proposons d'étudier l'utilisation de méthodes d'apprentissage profond sur des images/vidéos compressées, afin de réduire leurs besoins en ressources et de permettre un déploiement à grande échelle des solutions logicielles développées par Actemium. Plus spécifiquement, nous ciblons deux types de compressions (la compression d'images JPEG et la compression vidéo MPEG4 part-2) pour deux applications spécifiques : la détection d'objets et l'estimation du débit de flux routiers.

Dans un premier temps, nous nous concentrons sur la détection d'objets dans les images compressées JPEG. Du fait que l'algorithme JPEG compresse les images depuis une représentation *spatiale* en une représentation *fréquentielle* par blocs, le principal défi consiste à concevoir des modèles de détection capables d'estimer correctement la *position* des objets depuis cette nouvelle représentation. En utilisant des images compressées au format JPEG comme entrées, nous développons des architectures d'apprentissage profond de détection d'objets et démontrons une accélération de la vitesse de prédiction d'un facteur 1,7 tout en ne réduisant la performance de détection que de 5,5%. De plus, nous démontrons empiriquement que seule une partie des informations compressées, la composante de luminance, est nécessaire pour atteindre la précision des méthodes utilisant l'ensemble des informations contenues dans les images.

Nous abordons ensuite le problème de l'estimation du débit routier (nombre de véhicules/unité de temps) à partir de flux vidéo compressés MPEG4 part-2 provenant de caméras de surveillance de tunnels routiers. L'algorithme de compression vidéo MPEG4 part-2 utilise une représentation approximative du flux de pixels entre les images pour réduire la taille des données à encoder. Cette représentation semble donc pertinente pour estimer le débit de flux routiers tout en réduisant les besoins en resources de calcul et en mémoire. Nous proposons plusieurs architectures d'apprentissage profond de type *end-to-end* qui utilisent cette représentation comme entrée. En utilisant ces architectures, nous démontrons que la prédiction du débit routier à partir de flux vidéo compressés MPEG4 part-2 est possible tout en atteignant une meilleure précision par rapport à un modèle plus classique, basé sur les vidéos RGB, et permet, de plus, d'accélérer de façon impressionnante l'étape de prédiction ($\times 3200$). Enfin, les données d'entraînement pouvant être difficiles à obtenir en raison de contraintes industrielles, nous étudions la possibilité d'utiliser des méthodes d'adaptation de domaine pour transférer les modèles appris d'une caméra à une autre et nous fournissons une analyse approfondie des contraintes qui peuvent entraver un tel transfert.

Acknowledgements

Thanks all !

Contents

Acronyms

AC coefficient: Alternating Current coefficient. 20, 25, *Glossary:* AC coefficient

ANN: Artificial Neural Network. 31, 32, 37, *Glossary:* ANN

CNN: Convolutional Neural Network. 34, 36, *Glossary:* CNN

ConvLSTM: Convolutional Long Short-Term Memory. 35, 36, 83–85, 88, 93, 111, 126, *Glossary:* ConvLSTM

CTC: Connectionist Temporal Classification. I, 4, 8, 30, 41–43, 45, *Glossary:* CTC

DC coefficient: Direct Current coefficient. 20, 23, 25, 27, *Glossary:* DC coefficient

DCT: Discrete Cosine Transform. 15, 16, 19–23, 25, 57, 58, 61, 74, *Glossary:* DCT

DPS: Datapoints Per Second. 93, 110, 111, *Glossary:* DPS

FPS: Frames Per Second. 50, 53–56, 58, 62, 66–72, 77, 81, 87, 88, 93, 94, 103, 110, 111, 114, *Glossary:* FPS

GPU: Graphics Processing Unit. 33, 53, 55–57, 66, 67, 71, 72, 85, 88, 93, 96, 110, 114, *Glossary:* GPU

HNM: Hard Negative Mining. *Glossary:* HNM

I-VOP: Intra coded VOP. 24–27, *Glossary:* Intra coded VOP

IDCT: Inverse Discrete Cosine Transform. 16, 26, *Glossary:* IDCT

IoU: Intersection over Union. 39, 53, 59, *Glossary:* IoU

LSTM: Long Short-Term Memory. 35, 36, *Glossary:* LSTM

mAP: mean Average Precision. 40, 41, *Glossary:* MV

MV: Motion Vector. 24, 25, 28, 76, 79, 80, 83, 88, 94, 96, 98, 100, 101, 104, 107, 108, 110, 111, 114, 124, 126, *Glossary:* MV

NMS: Non-Maximum Suppression. 39, 55, *Glossary:* MV

P-VOP: Predictive coded VOP. 24–26, *Glossary:* Predictive coded VOP

RLE: Run Length Encoding. 16, 21–23, 61, *Glossary:* RLE

RNN: Recurrent Neural Network. 35–37, 84, 86, *Glossary:* RNN

RoI: Region of Interest. 38, 52, 53, 78, 81, *Glossary:* RoI

VO: Video Object. 24, *Glossary:* Video Object

VOP: Video Object Plane. 24–26, *Glossary:* Video Object Plane

Glossary

AC coefficient: An Alternating Current coefficient is a DCT coefficient for which the testing frequency is non zero. 20

ANN: An Artificial Neural Network is a mathematical model in the deep learning framework able to fit a target function. Its name comes from its architecture that supposedly mimics the human brain. 31

CNN: A Convolutional Neural Network is a network based on convolutional layers. 34

ConvLSTM: A Convolutional Long Short-Term Memory (ConvLSTM) network is a peculiar type of RNN introduced by Shi et al. [2015] to process sequences of images. 35

CTC: The Connectionist Temporal Classification is a method introduced by Graves et al. [2006] to learn classification on unsegmented sequences. 4, 42

DC coefficient: The Direct Current coefficient is the DCT coefficient for which the testing frequency is zero. 20

DCT: The Direct Cosine Transform is a transform aiming to extract the frequency information contained within a signal. 15

DPS: Datapoints Per Second. 93, 110

FPS: Frames Per Second. 50, 66, 103

GPU: Graphics Processing Unit. 33

IDCT: The Inverse Direct Cosine Transform (IDCT) is the inverse function of the DCT. It reconstructs a signal from its frequency components. 16

Intra coded VOP: An Intra coded VOP is a VOP that was coded without any reference to other VOP. 24

IoU: The Intersection over Union is a measure of how well objects are overlapping. 39, 53

LSTM: A Long Short-Term Memory (LSTM) network is a peculiar type of RNN introduced by Hochreiter and Schmidhuber [1997]. 35

MV: A Motion Vector is a vector indicating the position of the best matching macro-block in the previous reference frame with regard to the current macro-block. 24, 76, 100

NMS: The Non-Maximum suppression is a procedure used for object detection that aims to remove multiple overlapping detections. 39, 40, 55

Predictive coded VOP: A Predictive coded VOP is a VOP that was coded using a previous VOP as reference. 24

RLE: Run Length Encoding is a compression method that reduces the size of messages through a smart representation of series of repeating symbols. 16

RNN: A Recurrent Neural Network is a network used for sequence processing that recalls itself at each time step. It keeps an internal state to yield predictions based on previous inputs. 35

RoI: Region of Interest. 38, 52, 78

Video Object: A Video Object is used in the MPEG4 part-2 compression to represent a series of objects through time. Usually, for simplification purposes, a Video Object represents a series of frames through time. 24

Video Object Plane: A Video Object Plane is used in the MPEG4 part-2 compression to represent object at a given instant. Usually, for simplification purposes, a Video Object Plane represents a given frame. 24

Introduction

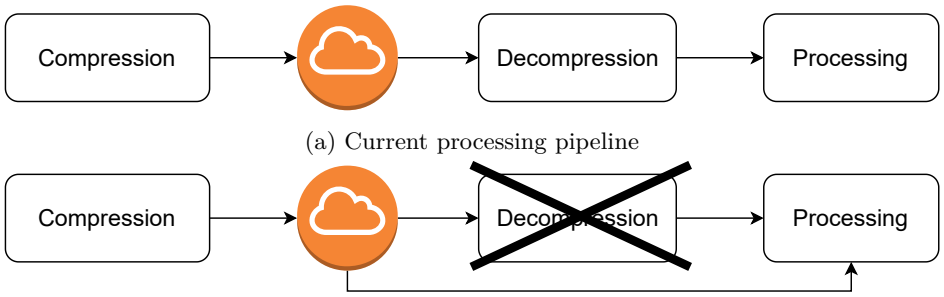
> " *The problem with quotes found on the internet is that they are often not true.* "
>
> Abraham Lincoln

Context and motivation

In recent years, the field of artificial intelligence has experienced a rapid and unprecedented growth. Problems that have been considered out of reach for many years can now be easily solved thanks to deep learning methods. In particular, deep learning has been intensively used in the field of computer vision. The craze started in 2012, when Krizhevsky et al. [2012] won by a large margin the ImageNet image classification challenge [Russakovsky et al., 2015]. Since then, a plethora of methods, mostly RGB-based, have been proposed, covering many image/video related applications. However, despite the apparent success of deep learning solutions, many companies are still struggling to integrate these new technologies into their products and decision-making procedures. The reasons for these difficulties are numerous (high data and computation resources requirements, explainability, etc.) and solving these issues is an active field for many industry companies.

Actemium Paris Transport is a subsidiary of Actemium working to develop systems for the monitoring of Paris' road tunnels. As of today, about 2000 cameras are deployed for the surveillance of the tunnels, and, in order to help the operators with the monitoring, semi-automatic surveillance systems are deployed. Such systems are critical as they help the operators initiate safety protocols in case of incidents. The current surveillance systems, and in particular the video processing modules, use methods heavily relying on heuristics. Although effective, these methods have several drawbacks. They are inclined to high rate of false alarms and may overload the operators with irrelevant alerts during rush hour. Moreover, due to the dependence of such methods on heuristics, their large scale deployment is a tedious task. As Actemium aims to develop solutions free from these limitations, they naturally seek to develop and test new solutions based on deep learning architectures. However, deep architectures require large datasets to train and high computation and memory resources when deployed. Given the scale of the surveillance systems, costs would grow to unbearable levels. Therefore, adaptations to the classical deep learning approaches must be considered to circumvent the problems. In particular, Actemium considers leveraging the compressed representation of the data, so far largely ignored by classical deep learning models, so as to reduce computation costs.

Onsite, video streams produced by the surveillance cameras constantly transit over the company closed internet network. As this network has limited bandwidth, all the video streams are heavily compressed before being transferred. Therefore, the overall processing pipeline of the automatic surveillance systems goes as follow: videos are compressed by onsite coders, transferred to the processing units, uncompressed and then processed (c.f Figure 1a). This processing scheme has the main drawback of not leveraging the compact compressed



(a) Current processing pipeline

(b) Targeted processing pipeline

Figure 1 – Current processing pipeline (a) and the one we pursue throughout the thesis (b)

representation of the road videos. It also induces heavy computation requirements. Indeed, not only is decompression costly computationally, but processing RGB images also requires a lot of computation resources in order to extract meaningful information.

Herein, the goal we pursue is to alleviate these downsides by designing prediction models, based on the compressed image and video representation, so as to skip the decompression step during processing (c.f Figure 1b). We also explore if the compressed representation of signals can be exploited to improve the processing stages. In particular, we target two specific computer vision tasks: object detection in compressed JPEG images and traffic flow rate estimation from compressed MPEG4 part-2 videos. Object detection is a critical task for Actemium as it allows to detect vehicles stopped in unauthorized areas and to raise alerts faster in case of an incident. As for flow rate estimation, the general surveillance of traffic allows to take preventive actions such as the recommendation of a secondary itinerary, so as to avoid congestions of the road network.

Data compression

Data compression is used extensively to avoid the saturation of the storage drives and internet networks. It has never been as important as in the recent years, due to the ever growing amount of data being exchanged and stored. As an example, in 2019, Snapchat users reportedly generated 527,760 photos *every minute*[1]. And, more recently, due to the lockdown, Netflix had to lower its streaming quality to reduce the strain on European internet service providers[2].

Data compression can be traced back to the late forties/early fifties, with the development of information theory by Claude Shannon [Shannon, 1948], Robert Fano [Fano, 1949] and David Huffman [Huffman, 1952]. Over the years, a multitude of compression formats have been proposed to cover an always larger range of uses (images, videos, sounds, etc.). For instance, for image compression, JPEG was proposed in 1992, JPEG2000 in 2000 and, more recently, WebP, created by Google, in 2018. Video compression followed a similar pace, with the H.26x family of compression, starting from H.261 in 1988 and currently at H.265, released in 2013 (H.262 in 1995, H.263, aka MPEG4 part-2, in 1999 and H.264 in 2003).

While targeting different usages, the image and video compression formats usually share common attributes. For instance, most of the image compression algorithms take advantage of either (or both) the limitations of the human eyes or spatial redundancies to reduce the size of the images. As for video, beyond spatial redundancies, the compression formats also leverage temporal redundancies and usually only encode the differences between subsequent frames.

[1] https://bernardmarr.com/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/

[2] https://www.bbc.com/news/technology-51968302

Deep learning and computer vison tasks

Deep learning is a method for automatic learning renown for its impressive results (AlphaGo[3], DeepL[4], etc.) and its need for data. Indeed, as the main strength of deep learning comes from its capacity to extract statistical properties from data and to use it to estimate target values, every ounce of added data improves the quality of learning. Due to its efficiency, deep learning has rapidly replaced older methods that required to carefully *handcraft* a complex processing pipeline, from the feature extractor to the estimation unit.

In particular, image and video processing have seen a surge of deep-learning-based solutions over the past years. The models used for image processing heavily rely on convolutions to extract information. Several convolutional deep learning architectures have been proposed to solve a large variety of computer vision tasks ; image classification [Krizhevsky et al., 2012, He et al., 2016, Simonyan and Zisserman, 2015], image segmentation [Long et al., 2015], object detection [Liu et al., 2016, Redmon et al., 2016, Ren et al., 2015], action recognition [Gowda et al., 2020], object tracking [Wojke et al., 2017], etc. However, while the emergence of such methods has been made possible by the gathering of large datasets, they are, in general, completely oblivious to the compressed representation of the images or videos.

Very recently, the combined usage of the data compressed representation and deep learning methods has seen a resurgence of interest so as to reduce both computation and bandwidth requirements. For instance, Gueguen et al. [2018], propose a system based on compressed JPEG images and deep learning for image classification. They show impressive results with speedup gains up to $\times 1.77$. Similarly, Chamain and Ding [2019] also manage to run classification on JPEG2000 compressed images. Such results are very encouraging and lean towards the effective learning of deep vision models on compressed images/videos. Yet, many challenges still lie ahead before such tools can be widely generalized.

Contributions

The main proposition of the thesis is to exploit the compressed version of signals to perform computer vision tasks. This proposition is declined amongst two main contributions, which cover two compressed signals, JPEG image compression and MPEG4 part-2 video compression. The first contribution addresses object detection in compressed JPEG images and is detailed as follows:

- We show that detection in the compressed domain can nearly reach detection performance of the RGB domain, and highlight its interest in settings where resources are scarce.
- We experimentally demonstrate that images contain large portions of unnecessary information for the task of object detection.

The second contribution is the estimation of the flow rate of moving objects on videos based on the compressed MPEG4 part-2 representation. We also introduce two new datasets for traffic flow estimation. This contribution is summarized as follows:

- We propose a new method that can be trained in an end-to-end fashion for flow rate estimation.
- We thoroughly study its behavior both on real and generated data.
- We demonstrate how domain adaptation can be used to overcome the lack of data and statistical shift inherent to industrial applications.

[3] https://fr.wikipedia.org/wiki/AlphaGo
[4] https://fr.wikipedia.org/wiki/DeepL

- We introduce a simulation dataset based on MNIST, that allows to generate video data according to targeted camera settings.
- We collect and annotate a dataset based on real surveillance cameras in road tunnels.

Publications

The contributions on object detection have been published in the following papers:

- Benjamin Deguerre, Clément Chatelain, and Gilles Gasso. *"Fast object detection in compressed jpeg images"*, in 2019 IEEE Intelligent Transportation Systems Conference (ITSC).
- Benjamin Deguerre, Clement Chatelain, and Gilles Gasso. *"Object detection in the DCT domain: is luminance the solution?"*, in 2020 25th International Conference on Pattern Recognition (ICPR).

Contributions related to video processing are in working progress.

Outline

The manuscript is divided into 5 chapters grouped into two parts. The first part focuses on the base knowledge required to understand the contributions. The second part focuses on the contributions. The content of the chapters is summarized as follows:

- Chapter 1 introduces the two compression formats, JPEG image compression and MPEG4 part-2 video compression.
- Chapter 2 reviews the core concept of deep learning, with a specific focus on object detection and the Connectionist Temporal Classification (CTC) loss (sequence segmentation learning).
- Chapter 3 tackles the task of object detection in compressed JPEG images. Multiple architectures addressing the specificities of the JPEG norm are proposed and we empirically demonstrate that the usual RGB input contains more information than required for the detection task.
- Chapter 4 addresses the task of flow rate estimation on compressed MPEG4 part-2 videos. Two types of deep architectures are proposed, one based on regression and one based on temporal classification (CTC). The models are thoroughly evaluated on a simulation dataset. Moreover, as data are often scarce in the industrial setting, we also experiment on domain adaptation so as to provide with better generalization capabilities for the proposed networks.
- Chapter 5 extends the work presented in chapter 4 towards real-world data. A new dataset for traffic flow estimation in tunnel videos is presented and the flow rate estimation methods proposed in chapter 4 are tested on this real-life dataset. We also extend the domain adaptation experiments, show limitations on the real-world data and explain their causes.
- Finally, the manuscript summarizes the main findings and sketches the perspectives for future work.

Introduction

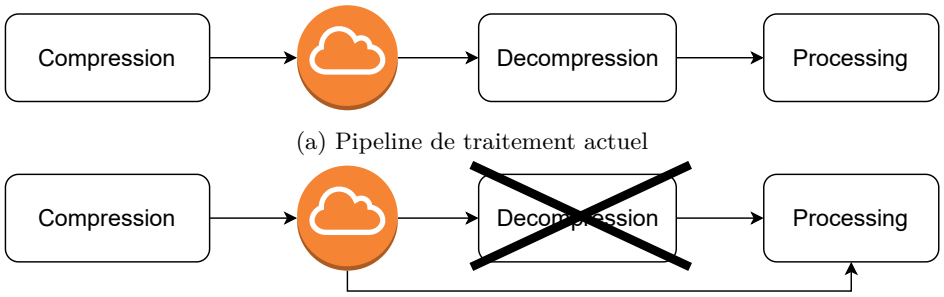
> “ *Le problème des citations trouvées sur Internet est qu'elles sont souvent fausses.* ”
>
> Abraham Lincoln

Contexte et motivation

Au cours de ces dernières années, le domaine de l'intelligence artificielle a connu une croissance rapide et sans précédent. Des problèmes considérés comme hors de portée pendant de nombreuses années peuvent désormais être résolus de manière élégante grâce aux méthodes d'apprentissage profond. En particulier, le domaine du traitement d'image a largement bénéficié des progrès récents. L'engouement pour l'apprentissage profond a commencé en 2012, lorsque Krizhevsky et al. [2012] a remporté haut la main le défi ImageNet de classification d'images [Russakovsky et al., 2015]. Dès lors, une pléthore de méthodes, principalement basées sur les images RGB, ont été proposées, couvrant de nombreuses applications liées à l'image/la vidéo. Cependant, malgré le succès apparent des solutions d'apprentissage profond, de nombreuses entreprises ont encore du mal à intégrer ces nouvelles technologies dans leurs produits et leurs chaines de traitement. Les raisons de ces difficultés sont nombreuses (besoins élevés en données et en ressources de calcul, difficultées à expliquer les décisions des réseaux neuronaux, etc.) et la résolution de ces problèmes est un domaine actif pour de nombreuses entreprises.

Actemium Paris Transport est une filiale d'Actemium qui travaille au développement de systèmes pour la surveillance des tunnels routiers de Paris. A ce jour, environ 2000 caméras sont déployées pour la surveillance des tunnels et, afin d'aider les opérateurs dans leur tâche, des systèmes de surveillance semi-automatiques sont déployés. Ces systèmes sont essentiels car ils aident les opérateurs à initier les protocoles de sécurité en cas d'incident. Les systèmes de surveillance actuels, et, en particulier, les modules de traitement vidéo, utilisent des méthodes basées sur des heuristiques. Bien qu'efficaces, ces méthodes présentent plusieurs inconvénients. Elles sont enclines à un taux élevé de fausses alarmes et peuvent surcharger les opérateurs avec des alertes non pertinentes aux heures de pointe. De plus, ces méthodes nécessitent d'être manuellement calibrées, ce qui rend leur déploiement à grande échelle fastidieux. Actemium, qui cherche à se libérer de ces limitations, se tourne naturellement vers le développement et le test de nouvelles solutions basées sur des architectures d'apprentissage profond. Cependant, l'utilisation des architectures existantes engendrerait des coûts très élevés lors de leur déploiement du fait de leurs besoins en ressources de calcul et en mémoire élevés. Par conséquent, Actemium cherche à modifier les approches classiques afin de contourner ce problème. En particulier, Actemium envisage de tirer parti de la représentation compressée des données, jusqu'ici largement ignorée par les modèles classiques d'apprentissage profond, afin de réduire les coûts en roussources de calcul.

Sur site, les flux vidéo produits par les caméras de surveillance transitent sur le réseau Intranet de l'entreprise. Comme ce réseau a une bande passante limitée, tous les flux vidéos



(a) Pipeline de traitement actuel

(b) Pipeline de traitement visé

Figure 2 – Pipeline de traitement actuel (a) et celui que nous cherchons à atteindre tout au long de la thèse (b)

sont fortement compressés avant d'être transférés. Par conséquent, le pipeline de traitement général des systèmes de surveillance automatique est le suivant : les vidéos sont compressées sur site par des codeurs, transférées vers les unités de traitement, décompressées puis traitées (cf. Figure 2a). Ce schéma de traitement présente le principal inconvénient de ne pas tirer parti de la représentation compressée des vidéos routières. Il implique également des calculs lourds. En effet, non seulement la décompression est coûteuse en termes de calcul, mais le traitement des images RGB nécessite également beaucoup de ressources afin d'extraire l'information présente dans les images.

Dans cette thèse, nous cherchons à contourner ces inconvénients en concevant des modèles de prédiction basés sur la représentation de l'image et de la vidéo compressées, afin d'éviter l'étape de décompression pendant le traitement (c.f Figure 1b). Nous cherchons également à savoir si la représentation compressée du signal peut être utilisée pour améliorer les modèles neuronaux. Plus particulièrement, nous ciblons deux tâches spécifiques du traitement d'image : la détection d'objets dans des images JPEG compressées et l'estimation du débit du trafic routier à partir de vidéos MPEG4 part-2 compressées. La détection d'objets est une tâche critique pour Actemium car elle permet de détecter les véhicules arrêtés dans des zones non autorisées et de déclencher les alertes plus rapidement en cas d'incident. Quant à l'estimation du débit, la surveillance générale du trafic permet de prendre des mesures préventives telles que la recommandation d'un itinéraire secondaire afin d'éviter les congestions sur le réseau routier.

Compression de données

La compression des données est largement utilisée afin d'éviter la saturation des espaces de stockage et du réseau Internet. Elle n'a jamais été aussi importante qu'au cours de ces dernières années en raison de la quantité toujours plus importante de données échangées et stockées. À titre d'exemple, en 2019, les utilisateurs de Snapchat ont généré 527 760 photos *chaque minute*[5]. Et, plus récemment, en raison du confinement, Netflix a dû diminuer la qualité de ses vidéos pour réduire la pression sur le réseau Internet européens[6].

L'étude de la compression des données remonte à la fin des années 40 et au début des années 50, avec le développement de la théorie de l'information par Claude Shannon (Shannon), Robert Fano (Fano) et David Huffman (Huffman). Au fil des ans, une multitude de formats de compression ont été proposés pour couvrir un éventail toujours plus large d'utilisations (images, vidéos, sons, etc.). Par exemple, pour la compression d'images, JPEG a été proposé en 1992, JPEG2000 en 2000 et, plus récemment, WebP, a été créé par Google, en 2018. La

[5] https://bernardmarr.com/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/

[6] https://www.bbc.com/news/technology-51968302

compression vidéo a suivi un schéma de développement similaire avec la famille de compression H.26x, depuis H.261 en 1988 et jusqu'à H.265, sorti en 2013 (H.262 en 1995, H.263 alias MPEG4 part-2 en 1999 et H.264 en 2003).

Bien qu'ils soient destinés à des usages différents, les formats de compression d'images et de vidéos ont généralement des caractéristiques communes. Par exemple, la plupart des algorithmes de compression d'image tirent parti des limites de l'œil humain et/ou des redondances spatiales pour réduire la taille des images. En ce qui concerne la vidéo, en plus des redondances spatiales, les formats de compression tirent également parti des redondances temporelles et n'encodent généralement que les différences entre les images successives.

Apprentissage profond et traîtement d'image

L'apprentissage profond est une méthode d'apprentissage automatique réputée pour ses résultats impressionnants (AlphaGo[7], DeepL[8], etc.) et ses besoins important en données. En effet, comme la principale force de l'apprentissage profond provient de sa capacité à extraire des propriétés statistiques des données et à utiliser ces dernières pour estimer des valeurs à prédire, chaque donnée ajoutée améliore la qualité de l'apprentissage. En raison de son efficacité, l'apprentissage profond a rapidement remplacé les anciennes méthodes qui nécessitaient de mettre en place un pipeline de traitement complexe, de l'extracteur de caractéristiques, jusqu'au module d'estimation.

Ces dernières années, les solutions basées sur l'apprentissage profond se sont multipliées dans le traitement des images et des vidéos. Les modèles utilisés pour le traitement d'images s'appuient fortement sur les convolutions pour extraire l'information des images. Plusieurs architectures convolutives d'apprentissage profond ont été proposées pour résoudre une grande variété de tâches de traîtement d'image : classification d'images [Krizhevsky et al., 2012, He et al., 2016, Simonyan and Zisserman, 2015], segmentation d'images [Long et al., 2015], détection d'objets [Liu et al., 2016, Redmon et al., 2016, Ren et al., 2015], reconnaissance d'actions [Gowda et al., 2020], suivi d'objets [Wojke et al., 2017], etc. L'émergence de ces méthodes a été rendue possible par la collecte de grands jeux de données. Cependant elles sont le plus souvent basées sur l'utilisation de la représentation RGB des images et vidéos et ne tirent généralement pas parti de la représentation compressée de ces dernières.

Très récemment, l'utilisation combinée de la représentation compressée des données et des méthodes d'apprentissage profond a connu un regain d'intérêt afin de réduire les besoins en calcul et en bande passante. Par exemple, Gueguen et al. [2018], propose un système basé sur des images JPEG compressées et l'apprentissage profond pour la classification d'images. Ils atteignent des résultats impressionnants avec des gains de vitesse allant jusqu'à $\times 1.77$. De même, Chamain and Ding [2019] réussissent également à faire de la classification sur des images compressées JPEG2000. Ces résultats sont très encourageants et vont dans le sens d'un apprentissage efficace de modèles de vision sur des images/vidéos compressées. Cependant, de nombreux défis restent à relever avant que de tels outils puissent être généralisés à l'ensemble des traîtements existant.

Contributions

L'objectif majeur de cette thèse est d'exploiter la version compressée des signaux pour réaliser des tâches de vision par ordinateur. Cette proposition se décline en deux contributions principales qui sont basées deux signaux compressés différents : la compression d'image JPEG et la compression vidéo MPEG4 part-2. La première contribution porte sur la détection d'objets dans des images JPEG compressées et est détaillée comme suit :

[7] https://fr.wikipedia.org/wiki/AlphaGo
[8] https://fr.wikipedia.org/wiki/DeepL

- Nous montrons que la détection dans le domaine compressé propose des résultats quasi équivalent à ceux obtenus avec des méthodes de détection dans le domaine RGB. De plus, nous soulignons l'intérêt de notre nouvelle méthode pour les cas où les ressources sont limitées.
- Nous démontrons expérimentalement que les images contiennent de grandes quantités d'informations inutiles pour la tâche de détection d'objets.

La deuxième contribution est l'estimation du débit d'objets dans des vidéos compressées au format MPEG4 part-2. Nous introduisons également deux nouveaux ensembles de données pour l'estimation du débit du trafic. Cette contribution est détaillée comme suit :

- Nous proposons une nouvelle méthode qui peut être entraînée de façon *end-to-end* pour l'estimation du débit de flux d'objets.
- Nous étudions de façon détaillé le comportement de cette nouvelle méthode sur des données réelles et générées.
- Nous démontrons comment l'adaptation au domaine peut être utilisée pour pallier au manque de données et le décalage statistique inhérent aux applications industrielles.
- Nous introduisons un jeu de données de simulation basé sur MNIST qui permet de générer des données vidéo en fonction de paramètres ciblés.
- Nous collectons et annotons un jeu de données basé sur des caméras de surveillance de tunnels routiers.

Publications

Les contributions sur la détection d'objets ont été publiées dans les articles suivants :

- Benjamin Deguerre, Clément Chatelain, et Gilles Gasso. *"Fast object detection in compressed jpeg images"*, in 2019 IEEE Intelligent Transportation Systems Conference (ITSC).
- Benjamin Deguerre, Clement Chatelain, et Gilles Gasso. *"Object detection in the DCT domain : is luminance the solution ?"*, in 2020 25th International Conference on Pattern Recognition (ICPR).

Les contributions relatives au traitement vidéo sont en cours de réalisation.

Organisation

Le manuscrit est divisé en 5 chapitres regroupés en deux parties. La première partie se concentre sur les connaissances de base nécessaires à la compréhension des contributions. La deuxième partie détaille les contributions. Le contenu des chapitres est résumé comme suit :

- Le chapitre 1 présente les deux formats de compression, la compression d'image JPEG et la compression vidéo MPEG4 part-2.
- Le chapitre 2 passe en revue les concepts de base de l'apprentissage profond, en mettant l'accent sur la détection des objets et la fonction objectif CTC (apprentissage de la segmentation des séquences).
- Le chapitre 3 aborde la détection d'objets dans les images JPEG compressées. Plusieurs architectures conçues pour la norme JPEG sont proposées et nous démontrons empiriquement que l'entrée RGB contient plus d'informations que nécessaire pour réaliser la tâche de détection.

- Le chapitre 4 aborde la tâche d'estimation du débit sur des vidéos MPEG4 part-2 compressées. Deux types d'architectures profondes sont proposés, l'une basée sur de la régression et l'autre sur de la classification temporelle (CTC). Les modèles sont évalués sur un jeu de données de simulation. De plus, comme ils est souvent difficile d'obtenir des données représentatives du domaine industriel, nous expérimentons l'adaptation de domaine afin de fournir de meilleures capacités de généralisation pour les réseaux proposés.
- Le chapitre 5 étend le travail présenté au chapitre 4 aux données de tunnels. Un nouveau jeu de données pour l'estimation du débit du trafic routier est présenté et les méthodes d'estimation du débit proposées au chapitre 4 sont testées sur ce jeu de données réelles. Nous étendons également les expériences d'adaptation de domaine, montrons les limitations sur les données industrielles et détaillons les causes de ces limitations.
- Enfin, le manuscrit résume les principaux résultats et esquisse les perspectives de travaux futurs.

Part I

Background and preliminaries

Chapter 1

Data compression

“ The suspense is terrible. I hope it will last. ”

Oscar Wilde

Contents

As an ever growing amount of data is being exchanged and processed on a daily basis, data compression has become one of the most important part of computer science. In particular, the rise of Deep Learning applications, that require tremendous amounts of data, would not have been so remarkable without efficient data compression methods. Yet, despite its importance, data compression is often overlooked as it is usually considered a mean rather than an end. As the contributions presented in this thesis are related to object detection and counting based on compressed information, this chapter aims to provide the reader with the basic tools to understand data compression. More specifically, we focus on image and video compression. Readers already familiar with these topics can skip this chapter.

Image compression aims to reduce the memory requirements for the storage and transfer of digital images. To improve the compression ratio when compared with classical compression algorithms, image compression algorithms usually take advantage of the limitations of the human eye (to remove imperceptible information) and spatial redundancies (to encode only once spatially duplicated data). As not all the compression algorithms use the same methods for compression, file formats are used to define the way images were encoded, and therefore the way to decode them. Image file formats can be divided into two main categories: vector formats and raster formats. Vector formats contain the geometric description of the images as group of connected points forming shapes. Due to the vectorial nature of this representation, images can be rendered at any desired display size without aliasing[1]. Vector images are, in some ways, the perfect representation to maximize the usage of spatial redundancies for compression. Indeed, as all points within a shape share the same properties they do not need to be encoded as long as the shape is correctly defined. Examples of vector image file formats are CGM or SVG. Unlike vector formats, raster images are defined as matrices of pixels and therefore do not cope well with zooms and other visual manipulations. While having obvious visual drawbacks, raster images are the most commonly used images. Indeed, when an image is recorded with a camera, it is hard to get a meaningful vectorial representation. Furthermore, the approximated image representation allows for a stronger and possibly lossy compression[2]. An example of strong lossy compression is given in Figure 1.1. The reconstructed image on the right loosely resembles the left image as the loss of information was extremely strong. Examples of raster image file formats are GIF, PNG, WebP, JPEG, JPEG 2000.

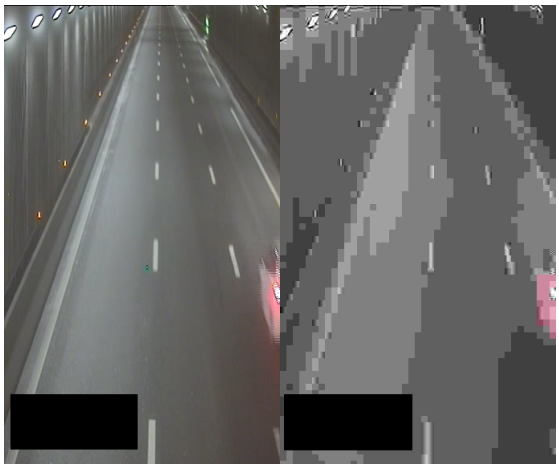


Figure 1.1 – Example of two different levels of compression. On the left, the image was lightly compressed, whereas on the right, the image is more strongly compressed. The strong compression results in loss of visual quality but also reduces the size of the image — from 235,8 ko to 2,2 ko.

[1] Aliasing is two different signals becoming indistinguishable. It can be seen, for instance, in images when diagonals take a stair-stepped appearance.

[2] A lossy compression is a compression that will lose information and therefore will not allow to perfectly reconstruct the original image.

As for images, video compression relies on the smart usage of spatial redundancies as well as the limitations of the human eye to reduce the size of the data to encode. However, unlike images, video compression algorithms also rely on temporal redundancies to further increase the compression ratio. Because various algorithms for video compression were developed over the years, compressed video data is usually stored in a video file format. A video file format is usually made of a container, a video coding format and an audio coding format. The container contains various metadata and is used to encapsulate the video data written in a specific video coding format alongside the audio data written in an audio coding format. Depending on the file format, multiple combinations of containers and coding formats can be used. It is therefore possible to have multiple files with identical extensions (i.e .mp4) that were encoded using completely different encoding specifications (i.e H.264, MPEG4 part-2, MPEG-2, MPEG-1). While this flexibility brings many advantages for the development of applications, it poses the problem of generalization for methods that are developed to take advantage of a given coding format.

In the thesis manuscript, we are interested in understanding image and video coding formats in order to bypass part of the decompression process for object detection and counting. As the data sent by the tunnels' cameras to the supervisors is encoded in either JPEG or MPEG4 part-2, the rest of this chapter focuses on these two compression methods and is divided as follow: first the JPEG compression algorithm is explained, and then the MPEG4 part-2 video compression algorithm is detailed.

1.1 JPEG image compression

JPEG is a compression norm introduced in 1992 by the Joint Photographic Expert Group[3]. Although the JPEG compression theoretically defines both a lossy (based on the Discrete Cosine Transform (DCT)) and a lossless version (that does not use the DCT for compression), usually, the JPEG compression refers to the lossy encoding process. Hereafter, we solely focus on the lossy version of the norm and leave aside its lossless form.

The JPEG norm defines multiple encoding processes (baseline, extended, ...) each of which defines parameters for a finer control of the related compression algorithms. To specify the process and parameters used for the encoding of a given image, the norm also defines a file format: the JPEG Interchange Format (JIF). However, this file format is rarely used on its own[4] due to the difficulties to implement every aspect of the norm and due to shortcomings that would not ensure identical rendering between decoders (for instance lack of color space definition). Therefore, extensions of the JIF were defined to reduce the range of encodings for JPEG images. As of today, the two main variants of the format in use are the JPEG File Interchange Format (JFIF, 1992) and the EXchangeable Image File format (EXIF, 1995). Because of that and as the RFC 2046[5] specifies that images transmitted over the internet should be JFIF compliant, we consider the JPEG norm to always follow the JPEG/JFIF specifications.

The rest of the section details the compression steps of the JPEG format. First an overview of the compression pipeline is given, then, each of the compression steps are more thoroughly explained.

[3] https://jpeg.org/jpeg/

[4] https://datatracker.ietf.org/doc/html/rfc2046

[5] Request For Comments (RFC) are documents describing aspects of technical internet specifications. The RFC 2046 aims to redefine the format of messages to allow for various body type (text, image ...).

1.1.1 Overview of the JPEG compression

The JPEG/JFIF encoding process can be divided into five main steps[6]: RGB to YC_bC_r change of color space, sub-sampling, block DCT computation, quantization and entropy coding/Run Length Encoding (RLE). Although algorithmically independent, these steps are semantically linked as they were designed to work together in order to improve image compression through loss of information without loss of visual quality. The whole compression pipeline is detailed in Figure 1.2, and the compression steps related to one another are enclosed in red boxes. The first two steps (RGB to YC_bC_r change of color space and sub-sampling) are designed to take advantage of the varying sensitivity of the human eye to color components to reduce the size of the image to encode. The last three steps (block DCT, quantization and entropy coding/RLE) exploit the differences in the human eye sensitivity to high and low frequencies and the relative homogeneity of natural images to reduce the size of the image to encode.

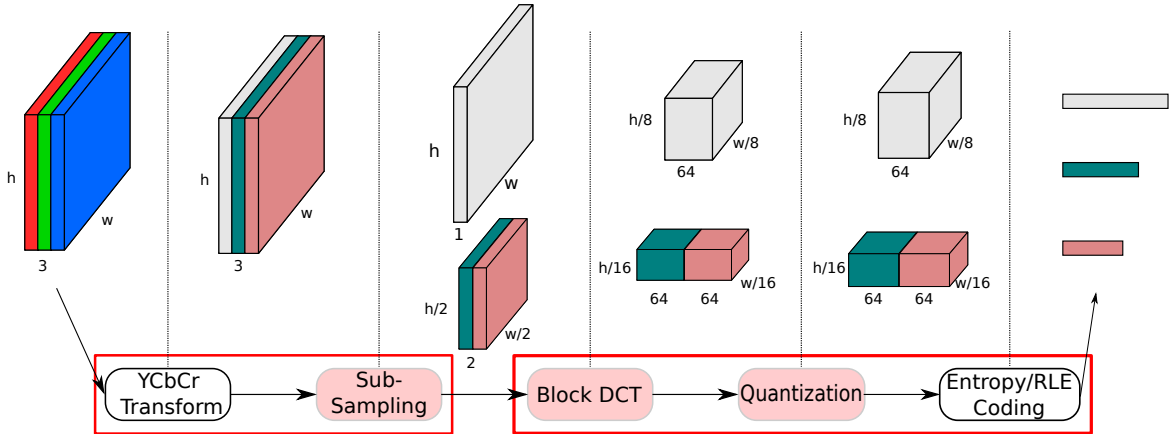


Figure 1.2 – The Full JPEG compression pipeline. The compression starts on the left with an RGB image and ends on the right with the entropy coded image. The compression steps in light red are lossy. The two red rectangles regroup the semantically linked compression operations. Image adapted from Gueguen et al. [2018].

It is to be noted that out of these five steps, three incur a non-reversible loss of information: sub-sampling, block DCT and quantization (blocks in light red in Figure 1.2). Both the sub-sampling and the quantization operations are lossy by design, they explicitly remove part of the visually unnecessary information. However, the block DCT operation is lossy due to algorithmic optimizations. To speed up the computation process using algorithms such as the ones developed by Chen et al. [1977] or Löffler et al. [1989][7], the JPEG norm allows the algorithm used for DCT/Inverse DCT (IDCT) computation some errors.

We now detail more thoroughly each step in the JPEG compression pipeline starting with the RGB to YC_bC_r change of color space.

1.1.2 YC_bC_r transform

The first step in the JPEG/JFIF compression pipeline is the RGB to YC_bC_r change of color space. This transformation aims to convert the image into a representation that will allow the next step (sub-sampling) to discard useless information. As the human eye is more sensitive to black and white than to color information [Lisa J. Croner, 2001, p. 204-205], using a smart color representation of the image to encode can help improve the compression ratio without

[6]For the sake of simplicity, algorithmic tricks such as shifts, encoding order and others are not detailed here. The interested reader can refer to the norm for more details.

[7]These references are given as examples of fast DCT algorithms, however, there is no guaranty that encoders do use these specific methods.

deteriorating the visual quality of the image. The YC_bC_r color representation separates the image visual information into three components: Y (the luminance[8]) which contains the black and white information of the image and C_bC_r (the chrominance) which contain the color information of the image. It is therefore possible to use this representation to decrease the size of the image (in the C_bC_r components) without reducing its visual quality. This is shown in Figure 1.3, where various sampling factors are applied to the color components of an image without loss of visual quality.

(a) Original image

(b) Image with a 4:2:0 sub-sampling

(c) Image with a 4:1:0 sub-sampling

Figure 1.3 – The same image at various sampling rates in the YC_bC_r domain. Although data is missing in the chroma components, the image is not visually altered.

As defined per the JPEG/JFIF specification, the linear formula to convert an image from the RGB to the YC_bC_r color space is given in Equation 1.1.

$$\begin{aligned} Y &= \min(\max(0, \text{round}(\ 0.299 \ * R + 0.587 \ * G + 0.114 \ * B \ )), 255) \\ C_b &= \min(\max(0, \text{round}(-0.1687 * R - 0.3313 * G + 0.5 \ * B + 128)), 255) \\ C_r &= \min(\max(0, \text{round}(\ 0.5 \ * R - 0.4187 * G - 0.0813 * B + 128)), 255) \end{aligned} \tag{1.1}$$

And the inverse operation used for decompression is given in Equation 1.2.

$$\begin{aligned} R &= \min(\max(0, \text{round}(Y \ + 1.402 \ * (C_r - 128))), 255) \\ G &= \min(\max(0, \text{round}(Y - 0.3441 * (C_b - 128) - 0.7141 * (C_r - 128))), 255) \\ B &= \min(\max(0, \text{round}(Y + 1.772 \ * (C_b - 128) \ )), 255) \end{aligned} \tag{1.2}$$

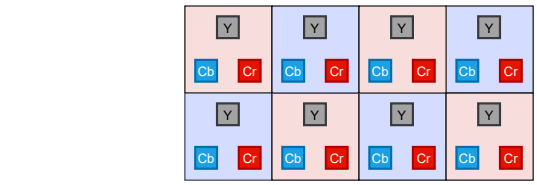
[8]It is important to note that the luminance is not used in its Physics meaning (that defines it as the luminous intensity per unit area of light travelling in a given direction) but as black and white information.

1.1.3 Sub-Sampling

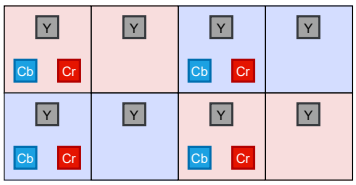
Following the YC_bC_r change of space, the next step in the JPEG compression is the sub-sampling operation. Sub-sampling consists in removing part of a signal at regular interval. For image processing, the sub-sampling operation is usually defined using a grid of 8 pixels (4 columns and 2 lines). Three numbers J:a:b are then used to specify which pixels should be kept:

- J represents the number of luminance samples to keep per line,
- a is the number of chrominance samples to keep on the first line,
- and b indicates the number of chrominance samples to keep on the second line.

The three most common sub-sampling formats are: 4:4:4, 4:2:2 and 4:2:0. In the first case (4:4:4), no sub-sampling is applied and all the luminance and chrominance samples are kept (see Figure 1.4a). The second sub-sampling format (4:2:2) is primarily used for video-related applications. In this format, the C_b and C_r components are sub-sampled by a factor of two in the horizontal dimension (see Figure 1.4b). The last format (4:2:0) is the most commonly used format. The color components are sub-sampled by a factor of two in both the horizontal and vertical dimensions (see Figure 1.4c).



(a) No sub-sampling



(b) Sub-sampling 4:2:2

(c) Sub-sampling 4:2:0

Figure 1.4 – Examples of various sub-sampling formats. For each format, the 4 luminance components are kept on the first and second lines (**4**:x:x). In a), no sub-sampling is applied. In b), 2 chrominance components are kept on the first (4:**2**:2) and second (4:2:**2**) lines. And in c), 2 chrominance components are kept on the first line (4:**2**:0) and 0 on the second (4:2:**0**). Images were modified from wikipedia.

It is to be noted that JPEG/JFIF does not specify how the sub-sampling operation should be performed. In particular, the specification, while recommending the usage of anti-aliasing filters[9], does not specify which one to use. This means that a given image could be encoded differently by two different encoders. Still, given the small differences this would produce, we consider the subsampling operation to be similar amongst the encoders. Furthermore, the JPEG norm does not define which component should or should not be sub-sampled. This means that, in theory, any of the Y, C_b or C_r components could be sub-sampled. However, the specification strongly discourages the usage of any sub-sampling format other than 4:2:0 and in practice, most of the images compressed in JPEG do use this specific sub-sampling format. Therefore, for the rest of this document, we assume a compressed JPEG image to always use the 4:2:0 sub-sampling format.

[9] Anti-aliasing filters aim to restrict the bandwidth of a signal to prevent aliasing (different frequencies becoming indistinguishable). Typical anti-aliasing filters are low-pass filters that remove frequencies above the Nyquist frequency (one-half of the sampling rate).

1.1.4 Block Discrete Cosine Transform (DCT)

The next step in the compression pipeline is the usage of the block DCT operation. This operation was designed to allow the subsequent compression steps to take advantage of two facts: first, small blocks of pixels are likely to have pixels of similar values (especially in natural images) and second, the human eye is less sensitive to higher frequencies in images. The latter can intuitively be visualized with a test image introduced by Campbell and Robson [1968] and shown in Figure 1.5. In this figure, the frequency of the luminance bands increases on the horizontal axis and the contrast decreases vertically. As the contrast decreases, our eyes have more trouble distinguishing the variations in the high frequency patterns than in the middle frequency patterns. Therefore, with a correct separation of the high and low frequencies, one can reduce the size of the images to encode while limiting the impact on the final rendering by discarding part of the high frequency information.

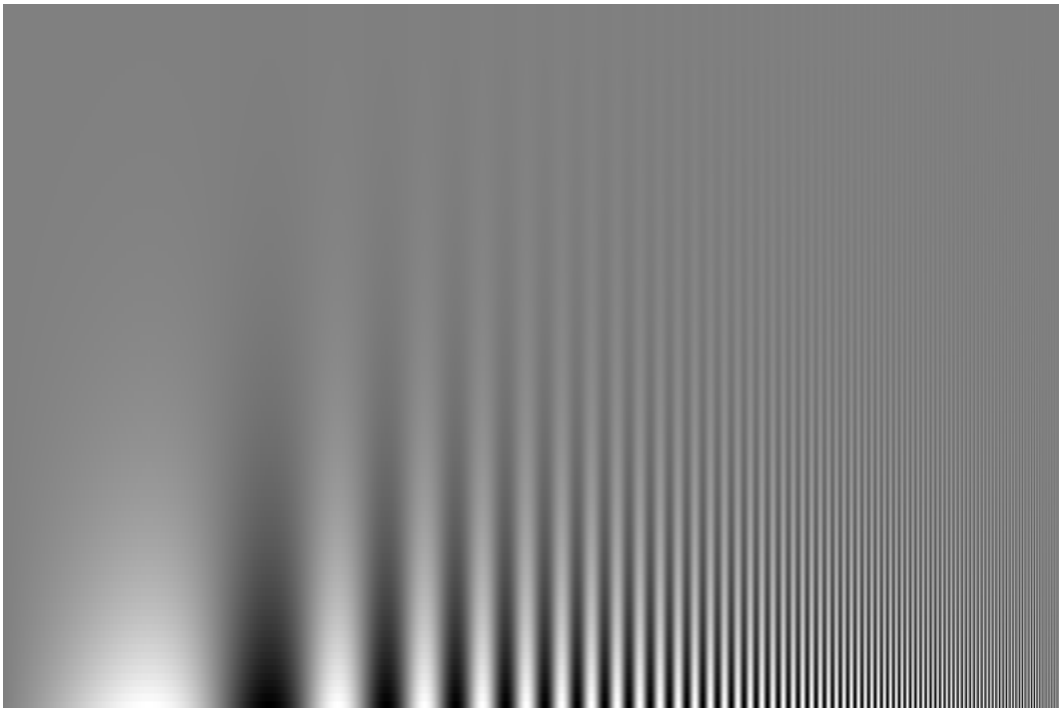


Figure 1.5 – Visual test chart for the accuracy of the human eye given changing frequencies. The frequency of the luminance bands increases on the horizontal axis and the contrast decreases vertically. Depending on the contrast and frequency values, our eyes are not able to see the alternating bands.

More specifically, the image to be JPEG compressed, after being converted to the YC_bC_r color space and sub-sampled, is divided into blocks of size (8×8). Then for each block, the 2D DCT is applied to extract the information at various frequencies in the blocks of pixels. The 2D DCT is based on the combination of two DCT-II (which is 1D):

$$S_v = \sum_{y=0}^{N-1} s_y \cos \frac{(2y+1)v\pi}{2N}, \tag{1.3}$$

where N is the number samples in the signal, $v \in [0, N-1]$ represents the frequency being tested against and s_y is the sample at position y in the signal. Each of the S_v expresses the amount of information at frequency v present in the signal. The 2D DCT operation on block of size 8×8 is then defined as:

$$S_{uv} = \frac{1}{4} C_u C_v \sum_{x=0}^{7} \sum_{y=0}^{7} s_{yx} \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16} \tag{1.4}$$

where,

$$C_u, C_v = \begin{cases} \frac{1}{\sqrt{2}}, & \text{for } u, v = 0 \\ 1, & \text{otherwise,} \end{cases}$$

and, with S_{uv} the DCT coefficient at position u, v in a given block, s_{yx} the pixel value at position x, y in a given block and C_u, C_v normalization coefficients. Usually, the coefficient at position $(0, 0)$ is referred to as the DC coefficient and the others as AC coefficients. Equation 1.4 can be further re-written to highlight the combined usage of two DCT-II as follows:

$$S_{uv} = \frac{1}{4} C_u C_v \sum_{x=0}^{7} \left[\sum_{y=0}^{7} s_{yx} \cos \frac{(2y+1)v\pi}{16} \right] \cos \frac{(2x+1)u\pi}{16} \tag{1.5}$$

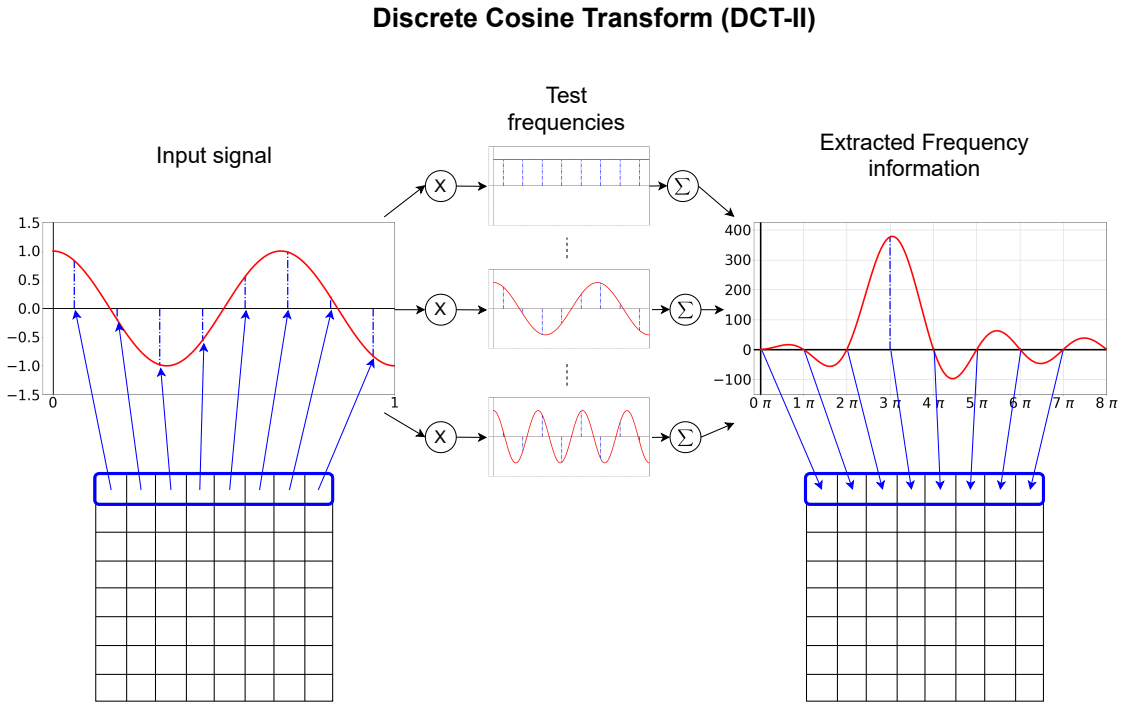


Figure 1.6 – Illustration of the DCT-II operation on a block of pixels. The pixel values are taken row by row (left) and multiplied by the test frequency values. Then the frequency information is stored in a row from low to high (left to right).

Intuitively, the inner part of Equation 1.5 will, for each row of the block to encode, apply the DCT-II operation and extract the frequency components of the signal and store them from low to high (from left to right), as illustrated in Figure 1.6. As a given block is expected to have pixels of similar values, information will, at this point, likely be concentrated in the low frequency coefficients and, for each DCT row of the block, be similar to one another. Then, the outer part of Equation 1.5 will, for each column (representing the information at a given frequency for each line), perform once more the DCT-II operation and extract frequency components of the previously extracted row-wise frequency components and store them from low to high (from top to bottom). This second DCT-II will therefore further concentrate most of the information in the top-left of the block in the low frequency pixels. The combination of the two operations will thus store the coefficients in a zig-zag pattern (from low to high) for the next compression steps. The overall procedure is shown in Figure 1.7.

It is important to see that due to the previous sampling operation, DCT transformed chrominance and luminance blocks do not represent an equivalent portion of the original image. Indeed, as the C_b and C_r components are sub-sampled by a factor of two, a luminance block contains information from 8×8 pixels, while the chrominance blocks includes information from 16×16 pixels.

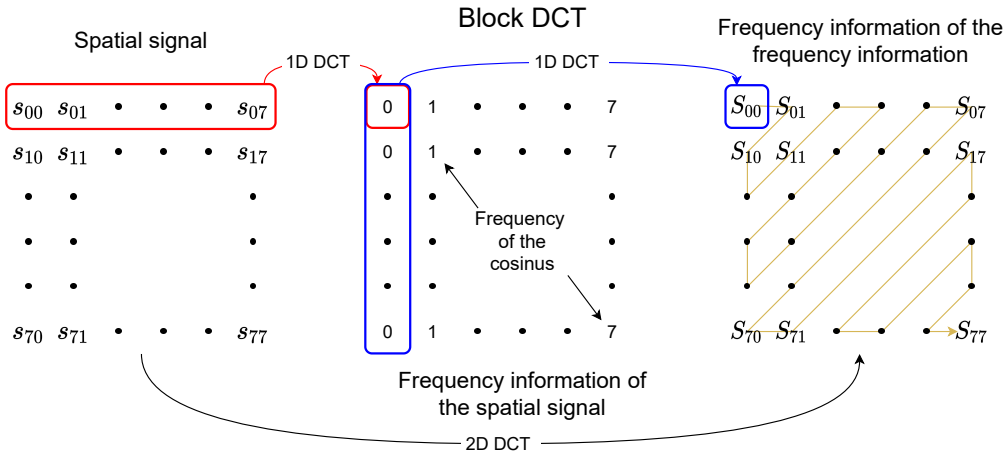


Figure 1.7 – Visual representation of the 2D DCT. First, each of the frequency components are extracted line per line and stored from left to right (low to high). Then, for each of the column representing the presence of a frequency in a line of pixels, the frequencies are again extracted. The combination of the two operations will have the frequencies stored in a zig-zag pattern from low to high.

Finally, notice that knowing DCT coefficient S_{uv}, one can retrieved back pixel values s_{yx} for a given block using the relation:

$$s_{yx} = \frac{1}{4}\sum_{u=0}^{7}\sum_{v=0}^{7} C_u C_v S_{vu} \cos\frac{(2x+1)u\pi}{16} \cos\frac{(2y+1)v\pi}{16} \tag{1.6}$$

Equation 1.6 is utilized at decoding stage of a JPEG image.

1.1.5 Quantization

After converting the image from the spatial domain to the frequency domain, the quantization step can be applied. The operation aims to filter part of the high frequencies in DCT blocks by taking advantage of the separation of the low and high frequencies. Quantization is applied using up to 2 quantization tables that are defined in the headers of JPEG files (as specified in the norm). A quantization table is a matrix of size (8×8) used to perform the following quantization operation:

$$Sq_{vu} = \text{round}\left(\frac{S_{vu}}{Q_{vu}}\right) \tag{1.7}$$

where $u, v \in [0, 7]$, Q_{vu} is the quantization coefficient at position (v, u) in the quantization table, S_{vu} is the pixel at position (v, u) in the DCT block to be quantized and Sq_{vu} is the resulting quantized pixel at position (v, u).

The quantization increases the compression efficiency of the last step of the compression pipeline in two ways. First, by squishing the range of values within a pixel block, the quantization allows for a better compression with the entropy encoding (see subsection 1.1.6). Second, by setting high values for the quantization coefficients at high frequencies, the operation generates series of zeros that can then be efficiently encoded using RLE (see subsection 1.1.6). These two behaviors of Equation 1.7 are shown in Figure 1.8.

Finally, it is important to note the main limitation of the quantization operation. The quantization tables can variate across image files and between components of a given image. This variation limits the interpretation that can be given of pure quantized image representations as shown in Figure 1.9. In this example, two DCT blocks are encoded with two different quantization tables but yield the same representation output.

At the decoding stage, the quantization is removed using the following equation:

$$R_{vu} = Sq_{vu} \times Q_{vu} \tag{1.8}$$

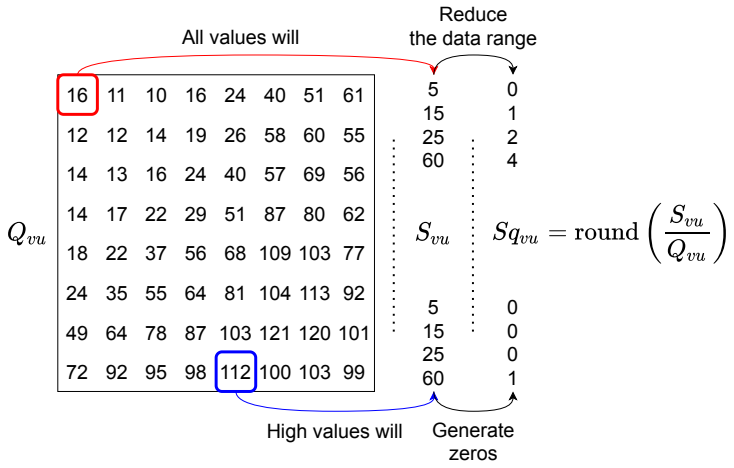


Figure 1.8 – Example of quantization table. When applied, all the coefficients will reduce the output range and for high quantization coefficients, will likely generate series of zeros.

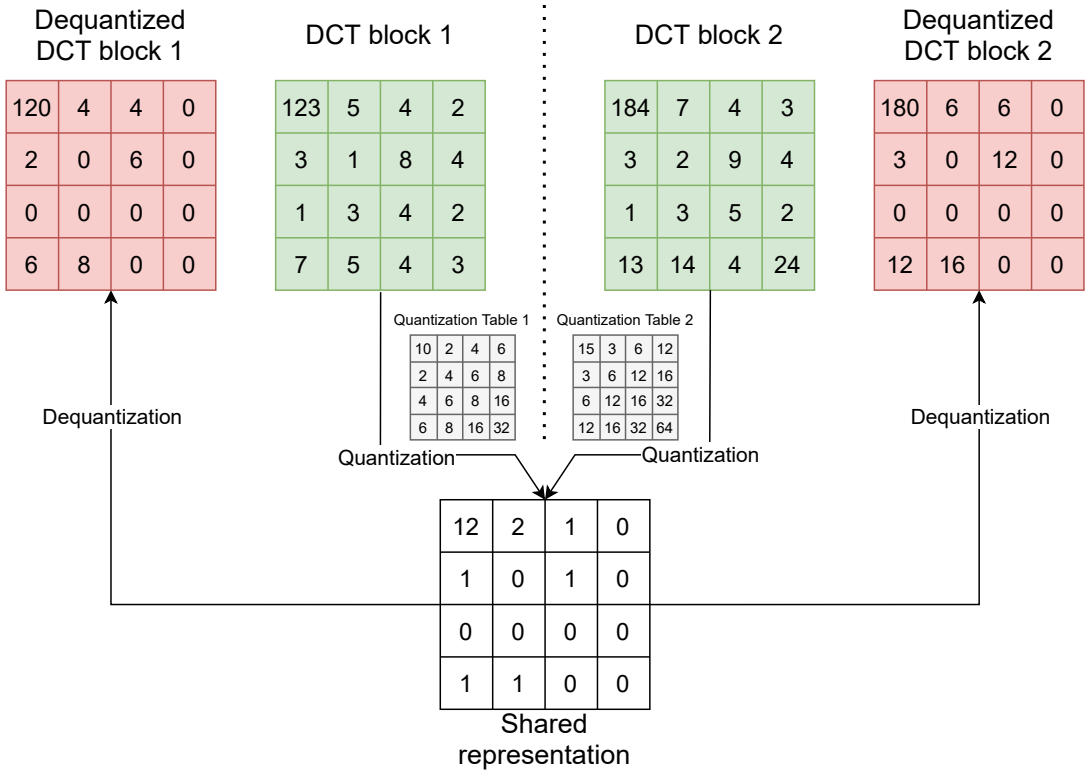


Figure 1.9 – Difficulties to interpret the quantized blocks as two different blocks may be identical while representing different input data.

where R_{vu} is the de-quantized pixel at position (v, u).

1.1.6 Entropy encoding/RLE

The last step in the JPEG compression pipeline is the Entropy encoding/RLE. It is this step that takes advantage of the previously generated representations to compress the image into its final representation. Each of the quantized DCT blocks are read through, and for each pixel of the blocks, two cases are to be distinguished: if the pixel is not equal to zero, it is encoded using Huffman coding, else, it is encoded using RLE.

Huffman coding [Huffman, 1952] is based on the probability of seeing a given symbol of a dictionary (in our case, all the possible values for the quantized coefficients). For each

symbol, a code of varying size is generated. The aim is to represent the most likely symbols with smaller codes to gain storage space. As the quantization operation squeezes the pixels values, it improves the final compression ratio by reducing the number of large codes to be used. The final compression is further improved by actually encoding the differences between consecutive DC coefficients rather than the actual coefficients as they can have high values. However, while it improves the compression ratio, this differential encoding also makes the interpretation of the encoded coefficients more difficult. Furthermore, the JPEG compression may use multiple lists of codes that are defined in multiple "Huffman tables". The usage of multiple Huffman tables limits the interpretation of the encoded blocks of data as similar entropy coded values may lead to completely different original blocks of data.

RLE is an operation which consists in transforming series of data into counts and values. For instance, the series "AAAAAAAAABBBBBBCCCCAAAAAAA" can be stored as "9A6B4C7A". Moreover, as in the case of JPEG compression, when encoding only one value with this method, only the size of the series can be used. As most of the information in blocks is contained in the low frequencies and as the quantization tables usually define high quantization values for high frequency pixels, it is expected to encounter large lists of following zeros in the high frequencies, therefore improving the final compression ratio.

1.1.7 Conclusion

This section has detailed the main steps of the lossy DCT based JPEG compression. These steps are RGB to YC_bC_r change of color space, sub-sampling, block DCT computation, quantization and entropy coding/RLE. The first two steps leverage the human visual system limits to reduce the amount of information to be stored. The last three steps take advantage of the difficulties of the human eye to differentiate information in the high frequencies to reduce the amount of information to be transmitted. The tunnel's camera can take snapshots of the road on request, such snapshots are encoded using the JPEG compression algorithm. Using this compression format, the tunnel images, of size $(352, 258, 3)$, i.e $352 \times 258 \times 3 \approx 272.5$ ko, can be compressed to an average size of 9.5 ko, which makes for a compression ratio of approximately 29. The next section details the MPEG4 part-2 video compression algorithm (also DCT based), which is used to encode the video data recorded by the cameras.

1.2 MPEG4 part-2 video compression

The MPEG4 norm (ISO/IEC 14496) aims to provide with a standard for the coding of audio-visual objects. It is composed of 33 parts, each of which covers a peculiar aspect of the specification (audio, video, etc.). Particularly, the part-2 of the MPEG4 norm (ISO/IEC 14496-2) focuses on video compression and aims to facilitate the access to visual objects within video streams. As the MPEG4 part-2 specification is aimed towards a large range of applications, it provides with a substantial variety of objects to be encoded within a stream: video objects, face and body animations objects and meshes objects. However, due to the difficulty to implement every aspects of the specification, profiles and levels were defined in order to limit the minimum requirements for a decoder to be considered compliant with the norm. A profile defines a subset of features that can be found in a coded video stream. A level defines a set of constraints imposed on parameters used in a profile. Decoders are therefore compliant with given combination of profiles and levels rather than with the whole MPEG4 part-2 norm. As this thesis targets video applications with industrial constraints, we choose to solely focus on the profile used to encode the video scenes of road tunnels: the "Simple Profile". This profile, as shown in the following subsections, is very similar to the JPEG norm in many aspects. Finally, it is important to notice that the MPEG4 part-2 specification does not enforce any constraints on the compression of a video flux and that only the decompression steps are given. Any flux that can be decoded by a decoder that is compliant with the norm is considered MPEG4 part-2 encoded. There is no regards for the

information that could have been lost during the compression process. Therefore, only the decompression steps are addressed herein.

The rest of the section is divided as follows: first an overview of the Simple Profile decompression pipeline is given, and then, each main decompression steps is detailed.

1.2.1 Simple Profile: General decoding pipeline

Motion Vectors and residual information:

The MPEG4 part 2 compression procedure relies on temporal redundancies to reduce the size of the coded video flux. The basic idea is to divide each of the video frames into blocks of pixels, and for each of the block to look for the best match in adjacent frames. Once the best match is found, the motion (difference between the position of the best match and the original block) and the residual information (difference between the two blocks) can be encoded. As the matching blocks are similar, the residual information is expected to be mostly zeros, therefore improving the compression ratio. For the Simple profile, a Motion Vector (MV) represent the motion information of a Macro Block of size (16×16). The whole matching method is shown in Figure 1.10 and a representation of a given RGB frame and the associated residual image and Motion Vector (MV)s is given in Figure 1.11.

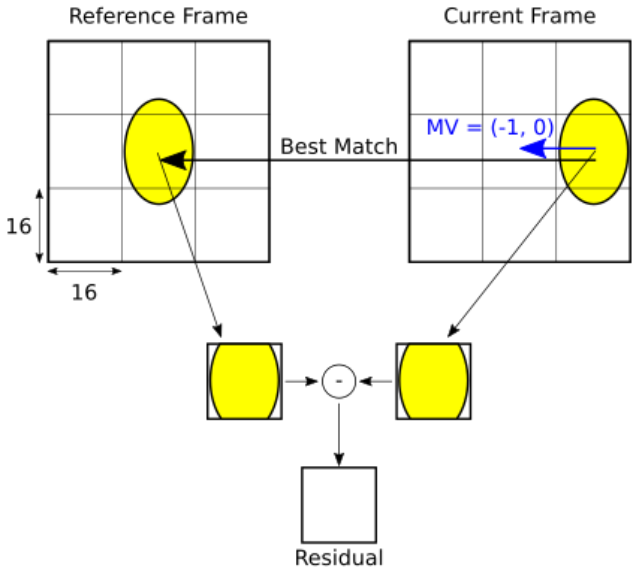


Figure 1.10 – Generation of the MVs and residual information. MVs are vectors of size 2, with one MV per block of image. Residual information is hold within matrices of the size of a block. In the example the Residual block would be full of 0 as the matching is perfect.

The Simple Profile makes use of one main object in the coded stream: the Video Object (VO). A VO is equivalent to a series of frames through time and a given frame at a given time is called a Video Object Plane (VOP). Not all the VOPs are coded using motion compensated predictions. Indeed, at least one fully encoded VOP is required to initialize the decoding process. Therefore, there are two types of VOPs that are defined in the specification[10]: Intra coded VOP (I-VOP, also called key frame) and the Predictive coded VOP (P-VOP, also called P-frame). I-VOPs are coded without reference to other pictures. They are the frames that the decoder will look for to "move" throughout or "join" a stream. P-VOPs are coded with reference to past I-VOP or P-VOP. As such VOPs use motion compensated coding, they are coded more efficiently than I-VOP. Thus, the number of I-VOPs within a coded flux corresponds to a trade off between the requirements for random access to the stream and the

[10]The specification actually defines two more types of VOP (B-VOP and S-VOP), however, as they are not used in the Simple Profile, they will not be detailed here.

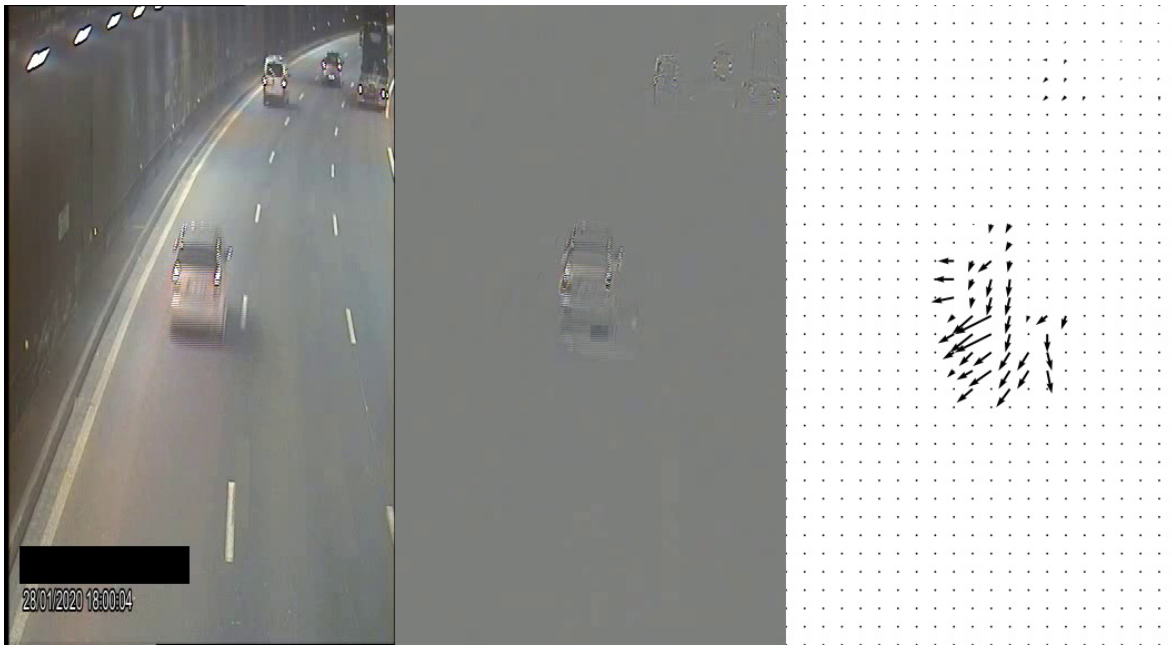


Figure 1.11 – Visualization of the original frame of a video stream alongside its compressed representation. From left to right are the original RGB frame, the residual image and the MV representation. We can see that only the vehicles generate data to be compressed as they are the only moving objects on screen. Motion vectors point in the opposite direction of the vehicles flow as they refer to previous frames.

compression ratio. Moreover, the number of I-VOPs also partly serves as an error resilience mechanism as they allow to re-initialize the decoding process.

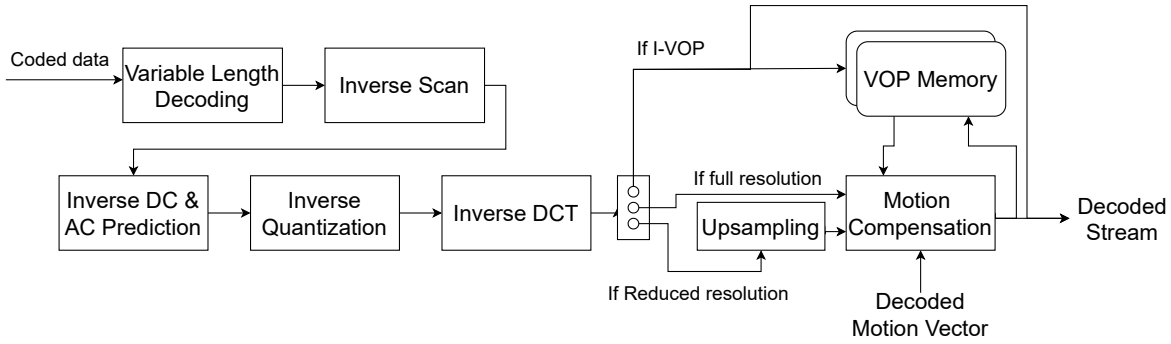


Figure 1.12 – Full VOP decoding pipeline.

Decoding scheme:

Both I-VOPs and P-VOPs go through a similar decompression pipeline, the main difference being the presence of the motion compensated prediction step. The whole decoding scheme is shown in Figure 1.12. Notice that while a MV represents the information of a Macro Block (16×16), the decompression steps apply to sub-blocks of size (8×8). First a coded block go through Variable Length Decoding step which is similar to the entropy/Run Length decoding step of the JPEG pipeline. Then the Inverse Scan Procedure is applied. This step aims to recompose the blocks of data from series of 64 numbers. Following this step, Inverse DC and AC prediction is applied. As for the JPEG compression, the coded flux actually contains the difference between the DC coefficients rather than the original ones. The main difference with the JPEG pipeline is that the first line (or column) of AC coefficients of a block can also be differentially encoded. Identical to the JPEG compression, the DCT coefficients are

then inverse quantized (following a different procedure). Then the IDCT is applied to the blocks of the VOP. After this step, there are two possibilities depending on the type of VOP being decoded. If the VOP is an I-VOP, then there are no more steps (except for operations such as color space transformation, data shifts, etc.) and the frame is stored for later motion compensated prediction and returned as a decoded frame. If the VOP is a P-VOP, then it is up-sampled (the Simple Profile implies a 4:2:0 sampling ratio) and motion compensated using the previously reconstructed VOP in order to generate the current frame. Then, the so reconstructed frame is process as for the I-VOPs.

The following subsections details the compression operations that significantly differ from the JPEG compression: inverse scan, inverse quantization and up-sampling.

1.2.2 Inverse Scan

The inverse scan aims to transform an array of pixels into a block of pixels. More specifically, it defines the position of each pixel in the entropy decoded array related to the quantized block. Where only one scan was usable for the JPEG compression (zig-zag), three versions are available for the MPEG4 part-2 compression. The three scan tables are shown in Figure 1.13.

0	1	2	3	10	11	12	13
4	5	8	9	17	16	15	14
6	7	19	18	26	27	28	29
20	21	24	25	30	31	32	33
22	23	34	35	42	43	44	45
36	37	40	41	46	47	48	49
38	39	50	51	56	57	58	59
52	53	54	55	60	61	62	63

Alternate-Horizontal scan

0	4	6	20	22	36	38	52
1	5	7	21	23	37	39	53
2	8	19	24	34	40	50	54
3	9	18	25	35	41	51	55
10	17	26	30	42	46	56	60
11	16	27	31	43	47	57	61
12	15	28	32	44	48	58	62
13	14	29	33	45	49	59	63

Alternate-Vertical scan

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

Zigzag scan

Figure 1.13 – Visual representation of the scan tables. The first two tables are transposes of one another. In these tables, the order of the numbers tends to follow lines (or column). In the last table, the ordering follows the same zig-zag pattern as for the JPEG compression.

Depending on the type of VOP, the scan tables that can be selected change. For P-VOP, only the zig-zag table can be selected. For the I-VOP, however, any of the three tables can apply. This possible variation in the coefficient order before the inverse scan operation implies some major limitations about the meaning of the data that can be extracted from the compressed flux before this point. Still, it can be noted that all the series share the same basic ordering from low to high frequency values.

1.2.3 Inverse Quantization

The MPEG4 part 2 compression norm proposes two inverse quantization operations. As we only consider the usage of the Simple Profile, only the second quantization method applies. It is based on a single weight, *quantiser_scale* and is defined in Equation 1.9.

$$|R_{vu}| = \begin{cases} 0 & \text{if } Sq_{vu} = 0 \\ (2 \times |Sq_{vu}| + 1) \times quantiser\_scale & \text{if } Sq_{vu} \neq 0, quantiser\_scale \text{ is odd,} \\ (2 \times |Sq_{vu}| + 1) \times quantiser\_scale - 1 & \text{if } Sq_{vu} \neq 0, quantiser\_scale \text{ is even.} \end{cases} \tag{1.9}$$

Here Sq_{vu} is the quantized coefficient of a given block of size 8×8 at position (u, v). The signed value of the inverse quantized coefficients R_{vu} is obtained using the following equation:

$$R_{vu} = \text{Sign}(Sq_{vu}) \times |R_{vu}|. \tag{1.10}$$

Furthermore, for the DC coefficients in the blocks of I-VOPs, inverse quantization is given by

$$|R_{00}| = dc\_scaler \times Sq_{00}, \tag{1.11}$$

where *dc_scaler* is a parameter that may be constant or indexed on *quantiser_scale* depending on the stream parameters. Although the quantization operation is solely defined by the parameter *quantiser_scale*, as this parameter may change, there is no guarantee that two identical quantized blocks across the stream represent the same decoded block of pixels.

1.2.4 Up-sampling

Contrary to the JPEG specification, the MPEG4 part-2 clearly defines the up-sampling operation. This written normalization has the advantage to ensure that any coded stream that is MPEG4 part-2 compliant, will present similar coded representation regarding color space.

As defined by the norm, there are two cases for the up-sampling operations: the first one, for inner pixels of the coded blocks, is shown in Figure 1.14 and, the second one, for outer pixels of the blocks, is illustrated in Figure 1.15.

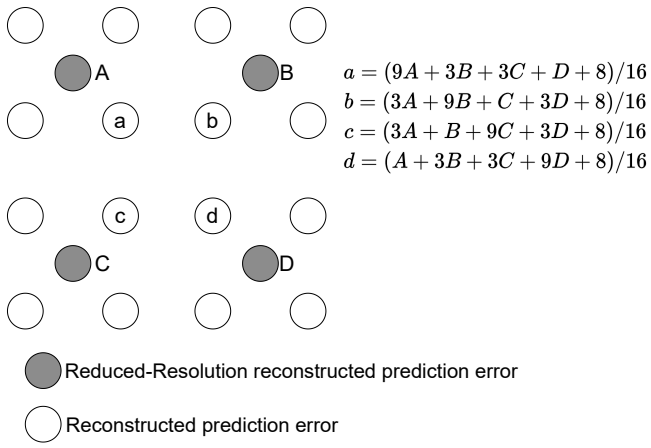


Figure 1.14 – Up-sampling processing for inner pixels of the coded block.

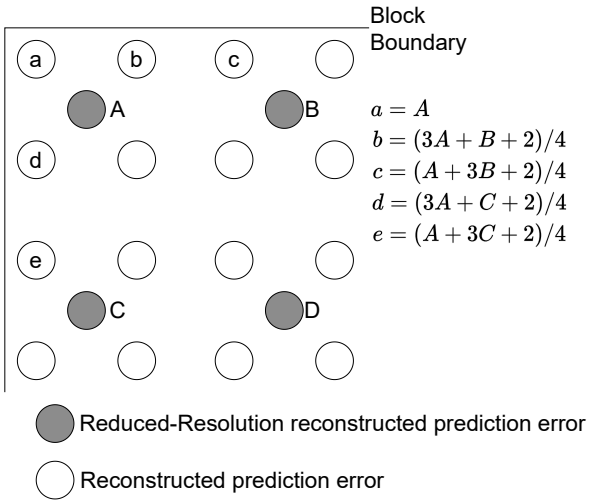


Figure 1.15 – Up-sampling processing for outer pixels of the coded block.

1.2.5 Conclusion

In this section we have detailed the MPEG4 part-2 decompression steps. Similarly to the JPEG norm, the MPEG4 part-2 norm takes advantage of the human visual system limits as well as spatial redundancies to reduce the size of the data to be encoded. However, unlike the JPEG compression, the MPEG4 part-2 norm also takes advantage of temporal redundancies to further increase the compression ratio. This temporal compression mainly seeks to solely encode the differences between frames in the form of MVs and residual data as illustrated in Figure 1.11. The tunnel's cameras are constantly recording the road for safety reasons and, due to bandwidth limitations, the generated videos are encoded in a MPEG4 part-2 compliant format. Using this compression format on the tunnel's recording gives a compression ratio that strongly variate between hours, from about 40 at 8 am, when the traffic is extremely dense, to 213 at 8 pm[11], when the vehicles are scarce on the road.

This chapter has provided with a deeper comprehension of data compression, in the light of JPEG and MPEG4 part-2 norms, that is mandatory to understand the work developed in this manuscript. The next chapter details the deep learning tools that are used throughout the contributions.

[11] These values were extracted from a small sample of videos.

Chapter 2

Deep Learning

“ I am so clever that sometimes I don’t understand a single word of what I am saying. ”

Oscar Wilde

Contents

In the previous chapter, we have detailed JPEG image compression and MPEG4 part-2 video compression. In the thesis we aim at applying the deep learning framework to such data for two main tasks: object detection and flow parameters estimation. This chapter introduces the base concepts of deep learning as well as the tools required to understand the contributions.

Before the deep learning era, data processing was usually done by carefully *handcrafting* a feature extractor so as to generate a relevant representation from input data that could then be used to estimate target values. While effective, this approach requires a deep understanding of the data nature as well as some intuitions about features that may reveal important for prediction. Deep learning aims to *learn* both the feature extractor and the predictor from data, limiting the needs for heuristics and feature engineering. Although enticing, deep learning has two major drawbacks that have hindered its use for many years: it has significant computational requirements and it necessitates a substantial data labelling effort. Nonetheless, thanks to impressive improvements in computation capabilities over the past years, as well as efforts from the scientific community to annotate and share large datasets, deep learning is now becoming a standard for pattern recognition in many fields of research.

As an example, object detection is a common problem for image and video processing that has been thoroughly studied over the years and is still an active domain of research. Object detection is usually cast as a regression/classification problem where the aim is to predict rectangular bounding boxes. Following this paradigm, a large number of deep learning architectures have been proposed throughout the years (R-CNN [Girshick et al., 2014], SSD [Liu et al., 2016], YOLO [Redmon et al., 2016], etc.), constantly improving the state of the art accuracy on various detection datasets. For instance, accuracy of object detection on the Pascal VOC challenge has almost quadruple since the start of the competition [Zou et al., 2019a]. This series of improvements can probably be related to the release of several carefully labelled datasets such as Pascal VOC [Everingham et al., 2010] or MS-COCO [Lin et al., 2014]. In these datasets, every object is annotated with its position, bounding box and class. This profusion of information, and more specifically the position, has been largely leveraged to help deep networks focus their learning on the objects within the images, leading to nowadays' impressive results.

However, annotating the position information of each object within the input is sometimes neither achievable in reasonable time nor desirable. Time achievability occurs for videos where each frame needs to be annotated. In such a case, completion of the full labelling would take a prohibitive amount of time. As for desirability, let's consider audio recordings. While it is simple to provide with the caption of each recording, their segmentations are much more complicated to obtain. As we speak in a continuous manner, pinpointing the exact moment of transition between two syllables is an arduous, if not impossible, task. For such cases, typical deep learning methods that leverage the position of signal constituents to focus their learning are usually voided. This issue has been, at least partially, addressed with the introduction of the CTC paradigm by Graves et al. [2006]. The CTC loss is used to jointly learn the recognition and the position of objects along a given dimension when only their ordering is known. This makes the CTC framework a very interesting tool for practical applications, where precise labelling may be complicated to obtain for time, cost and data quality reasons.

Hereafter, we begin by introducing the core of deep learning in order to provide with a quick reminder on basic notions, as well as a focus on specific parts that will be used throughout the contributions. Then, we focus on the RGB-based object detection (used in chapter 3 and chapter 5), detail its general formulation and the typical existing architectures. Finally, we provide with a deeper explanation of the CTC loss that is used throughout chapter 4 and chapter 5.

2.1 Basics of deep learning

Deep learning is a subset of the machine learning methods based on the usage of Artificial Neural Networks (ANNs) which aims at learning patterns within data. Deep learning originated from various attempts to represent the human brain from a mathematical perspective (Mcculloch and Pitts [1943], Rosenblatt [1963], etc.). The underlying hope was that such representation would provide with powerful learning capabilities. While the biological accuracy is, in the end, arguable, ANNs have proven themselves to be powerful learning tools.

In the deep learning framework, the aim is to construct a *model* parametrized by weights (from a few thousands up to a trillion [Fedus et al., 2021]) in order to fit a given target function. The learning of the models is done through gradient descent and more specifically by using the backpropagation algorithm [Kelley, 1960]. Models are constructed by stacking *layers* and multiple types of layers were proposed to handle different types of inputs. For instance, convolutional layers [Fukushima, 1980] were introduced to process images. They provide with translational invariance, which is extremely useful for task such as object detection. For sequence processing, "memory" was introduced through the recurrent layers. Such layers process the input sequentially while keeping an internal state, the memory of previous inputs, to help future decisions.

This section aims to introduce in more details the deep learning scheme as well as specific layers used in our contributions. We begin by reviewing the core concepts of deep learning (formulation and training), then move on to convolutional layers and finally detail recurrent layers.

2.1.1 Artificial Neural Networks

Artificial Neural Networks are based on the usage of artificial neurons that were designed to loosely mimic the functioning of the biological human neuron. A human neuron receives inputs through axones and dendrites and, depending on the power of the electrical impulse received, will in turn generate an impulse to subsequent neurons. Such functioning is mathematically formalized [Mcculloch and Pitts, 1943] using the following equation:

$$\phi_{\mathbf{w}}(\mathbf{x}) = \varphi\left(\sum_{j=0}^{J} w_j x_j\right). \tag{2.1}$$

Each input signal x_j is multiplied by a dedicated weight w_j, the resulting signals are then summed and fed to an activation function φ that decides if the neuron should activate or not and propagate the signal to subsequent neurons. Any function can serve as activation function, as long as it is continuous and non-linear. Typical activation functions are shown in Figure 2.1. Stacked neurons make a layer and stacked layers a deep model[1] (inner layers are called hidden layers). Therefore, to fit this model representation, we modify Equation 2.1 to the following (this peculiar equation represents a model made of dense layers):

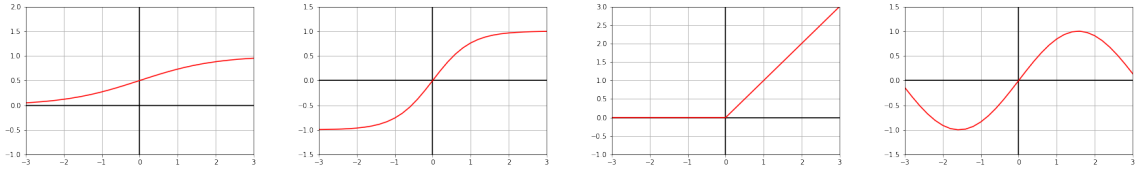
$$x_k^n = \phi_{\mathbf{w}_k^n}^n(\mathbf{x}^{n-1}) = \varphi^n\left(\mathbf{w}_k^{nT}\mathbf{x}^{n-1}\right), \tag{2.2}$$

where $\mathbf{w}_k^n$ is the vector of weights associated to the kth output neuron and n is the index of the layer. Usually, $\mathbf{x}^0$ is referred to as input, $\mathbf{x}^N$ (N being the number of layers in a model) as prediction or output and the remaining $\mathbf{x}^i$ ($i = 1, \ldots, N-1$) as *feature maps*.

2.1.2 Training ANNs

We have detailed the general architecture of the ANNs, we now review the training procedure used to estimate the target function. Formally, we aim to train a model $\mathcal{M} : \mathcal{X} \to \mathcal{Y}$, where

[1] The term deep learning actually refers to the depth of networks with multiple stacked layers



(a) sigmoid function (b) tanh function (c) ReLU function (d) sin function

Figure 2.1 – Examples of activation functions.

$\mathcal{X}$ is called the *input space* and $\mathcal{Y}$ the *output space*, in order to approximate a unknown target function. Models are trained to extract the statistical properties of the inputs in order to maximize the posterior probability $p(\mathbf{y}|\mathbf{x})$ of getting the correct $\mathbf{y} \in \mathcal{Y}$ given $\mathbf{x} \in \mathcal{X}$. Such maximization is done through the minimization of an objective function:

$$\min_{\mathbf{W}} \mathbb{E}_{(\mathbf{x},\mathbf{y})}[\mathcal{L}(\mathbf{y}, \mathcal{M}(\mathbf{x}, \mathbf{W}))], \tag{2.3}$$

where $\mathbb{E}$ is the expectation, $\mathcal{M}$ is the deep network, $\mathbf{W}$ are the weights of $\mathcal{M}$ to be set and $\mathcal{L}$ the loss function measuring how close $\mathcal{M}(\mathbf{x}, \mathbf{W})$ is to the true output. To carry out the minimization, gradient descent is usually used. The basic method to do such optimization is Stochastic Gradient Descent (SGD). SGD computes the average gradient

$$\Delta \mathbf{W}_{k-1} = \frac{1}{|B|} \sum_{i \in B} \Delta_{\mathbf{W}_{k-1}} \mathcal{L}(\mathbf{y}_i, \mathcal{M}(\mathbf{x}_i, \mathbf{W}_{k-1})), \tag{2.4}$$

where $(\mathbf{x}_i, \mathbf{y}_i)$ is the i-th element of the current batch B sampled from the training data, and updates the weights using the backpropagation algorithm as:

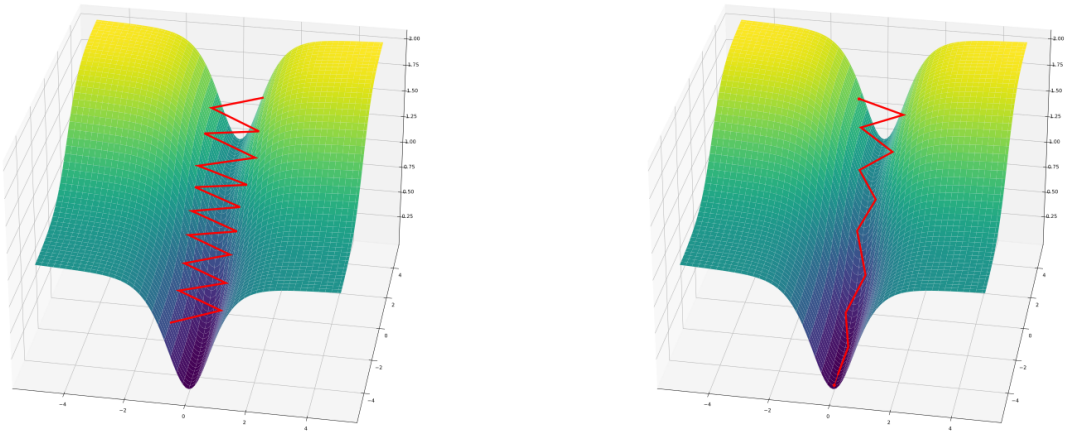
$$\mathbf{W}_k = \mathbf{W}_{k-1} - \eta \Delta \mathbf{W}_{k-1}, \tag{2.5}$$

where $\Delta \mathbf{W}_{k-1}$ is the gradient at iteration k and η is called the *learning rate*. For such optimization method, the speed and convergence towards a minima is highly dependent on the shape of the surface of the objective function w.r.t the weights. As shown in Figure 2.2a, in case of a ill-conditioned problem the optimization process may bounce back and forth, as the slope is very steep (therefore creating a strong gradient). Multiple update rules have been further developed to circumvent such problem: momentum [Qian, 1999] (Figure 2.2b), Adagrad [Duchi et al., 2011], RMSprop [Tieleman and Hinton, 2012], Adam [Kingma and Ba, 2015], to name the most common. Although each newer method is supposed to improve the optimization process when compared to older ones, which algorithm to use is highly application dependent and is still an open question.

2.1.3 Convolutional Neural Networks

In the previous subsection, we have detailed the basic architecture of ANNs as well as the training procedure used to optimize them. However, we only have considered one type of layer so far: the dense layer, where each output neuron is connected to each input through a weight. This layer, while functional is not best suited for all types of input. For instance, for image processing, each output neuron's weight would be connected to each pixel, therefore requiring a tremendous amount of weights. Furthermore, the dense layer is not invariant to translations, which is not a desirable property for tasks such as object detection.

A workaround to these problems is to use layers that operate in a sliding window fashion. As the window slides over the input image, weights are re-used at every target position. Therefore, translated inputs result in feature maps with translated activated neurons. Moreover,



(a) SGD optimization without momentum.

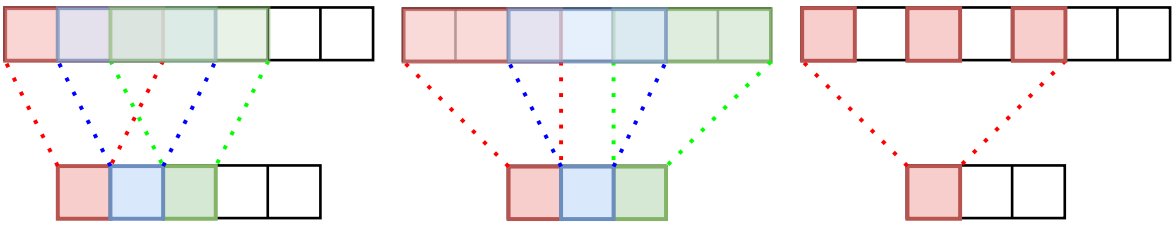
(b) SGD optimization with momentum

Figure 2.2 – SGD optimization with and without momentum. On the left, without momentum the optimization process will bounce between the ravine's slopes whereas, on the right, with momentum, optimization is smoother. Code to plot the curve was taken from the UvA DL course.

as the computations of each neuron within a feature map are independent, one can leverage the parallelism of GPUs to speed up processing. Layers with such behavior are called convolutional layers and their functioning is formalized as follows (1D convolutional layer):

$$x_i^n = \varphi^n \left(\sum_{m=0}^{M} w_m x_{i+m}^{n-1} + b \right), \tag{2.6}$$

where M is the size of the convolution kernel, and b is the bias parameter. It is to be noted that the input features x^{l-1} can be padded with zeros to provide with an output feature map x^l with identical shape. Moreover, stride and dilation can be applied to provide with more flexibility to the operation as illustrated in Figure 2.3.



(a) Classical convolution.

(b) Convolution with a stride of 2.

(c) Convolution with a dilation rate of 2.

Figure 2.3 – Convolution with different parameters. For clarity, the same filter has been represented with different colors. In a) classical convolution is applied at every position in the input. In b) the stride parameter is 2, therefore the convolution kernel skips one position after each computation. In c) the dilation parameter is 2 and only one input value out of two is considered at each position.

The formulation in Equation 2.6 allows for translational invariance. However, given a model with a single convolutional layer, the context (receptive field) a given neuron can gather is limited to the size of the kernel. Increasing context information requires to increase the size of the kernel and therefore the computation needed as well as the number of weights. A simple solution to this problem is to stack multiple convolutional layers. By doing so, the receptive field will naturally increase, while keeping the desired translational invariance property. This is illustrated in Figure 2.4.

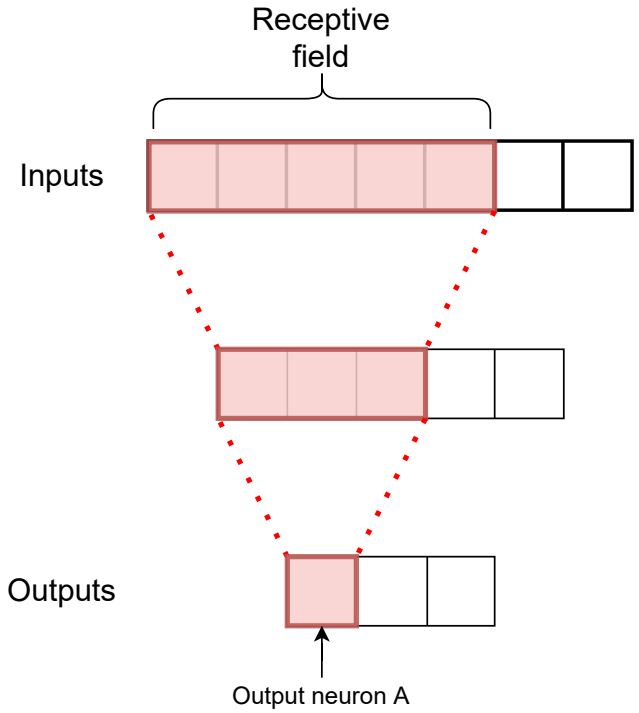


Figure 2.4 – The receptive field of neuron A on stacked convolutional layers with filters of size 3.

Finally, Equation 2.6 can easily be extended to any inputs of higher dimension. One simply needs to add a summation term for each added dimension. For instance, for 3D convolutions, Equation 2.6 becomes:

$$x_{lmn}^{n} = \varphi^{n}\left(\sum_{h=0}^{H}\sum_{w=0}^{W}\sum_{t=0}^{T} w_{wht}\, x_{l+h,m+w,n+t}^{n-1} + b\right). \tag{2.7}$$

This 3D convolution is represented in Figure 2.5.

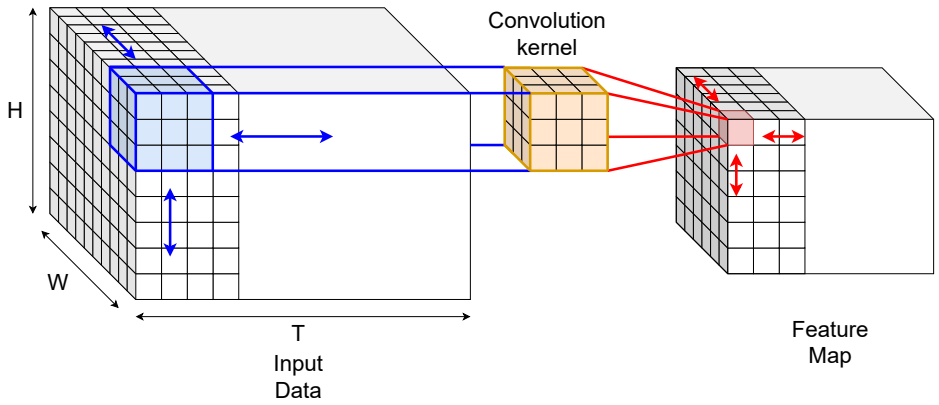


Figure 2.5 – Visual representation of the 3D Convolution operator.

Convolutional layers are extremely versatile tools. Thanks to their limited number of weights they help mitigate over-parametrization problems inherent to dense layers. Furthermore, they allow for easy parallelisation by design. As a result, convolutional layers are used nowadays throughout the scientific literature. For instance, 1D-CNNs have been used for automatic speech recognition [Kiranyaz et al., 2021], 2D-CNNs are the standard for image processing (Krizhevsky et al. [2012], He et al. [2016], Liu et al. [2016], Szegedy et al. [2016], etc.) and 3D-CNNs are used for medical applications [Singh et al., 2020].

2.1.4 Recurrent Neural Networks

While convolutional layers have interesting properties, they also have various limitations. For instance, they can only gather information up to the receptive field size. Although such behavior might be sufficient for entries of fixed shape, for entries of varying length, it would be better to manage to implement some form of memory that can gather a potentially unlimited amount of information. To do so, the Recurrent Neural Networks (RNNs) were introduced.

In essence, a RNN is based on the repetition of its architecture along the time axis. At each time step, a RNN layer takes in the current input and its previous hidden state (memory) and outputs the new hidden state (updated memory). This is expressed as:

$$\mathbf{h}_t = \varphi(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{b}), \tag{2.8}$$

where $\mathbf{W}$ and $\mathbf{U}$ are the inner weights used to respectively process the input and the previous hidden state, φ is the activation function, $\mathbf{b}$ is the vector of biases and $\mathbf{x}_t$ and $\mathbf{h}_t$ are respectively the input and hidden state at time t. The vanilla RNN is represented in Figure 2.6.

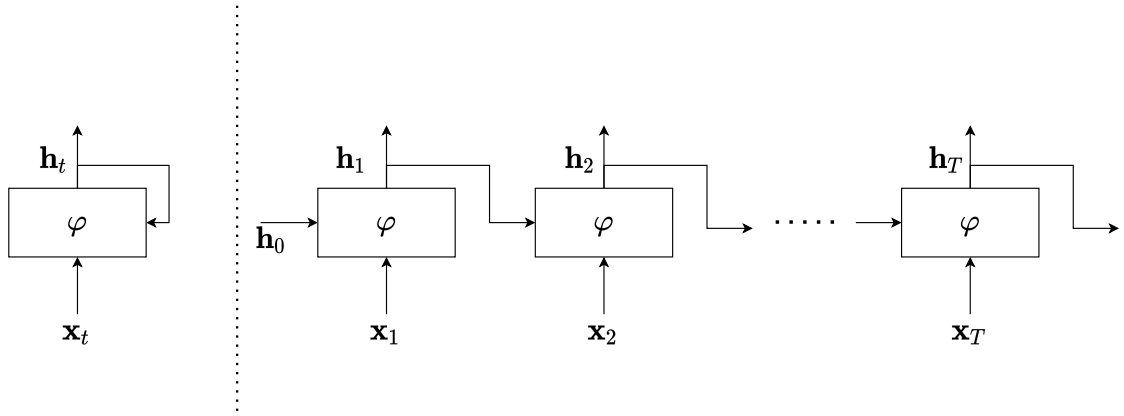


Figure 2.6 – Representation of a vanilla RNN network. For each input in the sequence, the layer predicts the output based on past information.

This basic formulation, however, is subject to the vanishing/exploding gradient problems. To correct this behavior, the Long Short-Term Memory (LSTM) network [Hochreiter and Schmidhuber, 1997] was introduced. Multiple variations of LSTMs exist, and we base the following description on Gers et al. [2000]. This architecture is made of a cell able to store information controlled thanks to three "gates" and two inner states (resp. input gate, output gate, forget gate, cell state and hidden state). They are given by:

$$\begin{aligned}
\mathbf{f}_t &= \sigma(\mathbf{W}_f\mathbf{x}_t + \mathbf{U}_f\mathbf{h}_{t-1} + \mathbf{b}_f),\\
\mathbf{i}_t &= \sigma(\mathbf{W}_i\mathbf{x}_t + \mathbf{U}_i\mathbf{h}_{t-1} + \mathbf{b}_i),\\
\mathbf{o}_t &= \sigma(\mathbf{W}_o\mathbf{x}_t + \mathbf{U}_o\mathbf{h}_{t-1} + \mathbf{b}_o),\\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \tanh(\mathbf{W}_c\mathbf{x}_t + \mathbf{U}_c\mathbf{h}_{t-1} + \mathbf{b}_c),\\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t),
\end{aligned} \tag{2.9}$$

where $\mathbf{f}_t$, $\mathbf{i}_t$, $\mathbf{o}_t$, $\mathbf{c}_t$, $\mathbf{h}_t$ are respectively the forget gate, input gate, output gate, cell state and hidden state, σ is the sigmoid function, $\mathbf{b}_x$ are the biases, the $\mathbf{W}_x$ and $\mathbf{U}_x$ are the weights, and $\odot$ is the Hadamard product. The architecture is further illustrated in Figure 2.7.

While extremely effective, the LSTM networks are not suited for sequence of images as they only process vectors. Therefore, to handle such data, one must first generate a meaningful vectorial representation and then apply the classical LSTM architecture. This representation has the major drawback of not using the spatial information for temporal processing. In order to correct this issue, Convolutional Long Short-Term Memory (ConvLSTM)

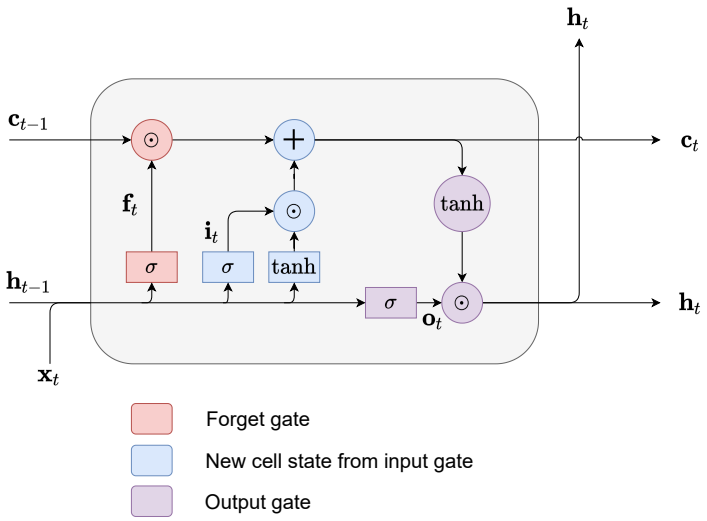


Figure 2.7 – Representation of a LSTM network. The forget gate selects previous information to be kept, then the input gate updates the cell state and, finally the output gate predicts the new output.

networks were introduced by Shi et al. [2015]. While Convolutional LSTMs keep the idea of input, forget and output gates, they modify how the functions are used:

$$
\begin{aligned}
\mathbf{I}_t &= \sigma(\mathbf{W}_{xi} * \mathbf{X}_t + \mathbf{W}_{hi} * \mathbf{H}_{t-1} + \mathbf{W}_{ci} \odot \mathbf{C}_{t-1} + \mathbf{b}_i) \\
\mathbf{F}_t &= \sigma(\mathbf{W}_{xf} * \mathbf{X}_t + \mathbf{W}_{hf} * \mathbf{H}_{t-1} + \mathbf{W}_{cf} \odot \mathbf{C}_{t-1} + \mathbf{b}_f) \\
\mathbf{C}_t &= \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \tanh(\mathbf{W}_{xc} * \mathbf{X}_t + \mathbf{W}_{hc} * \mathbf{H}_{t-1} + \mathbf{b}_c) \\
\mathbf{O}_t &= \sigma(\mathbf{W}_{xo} * \mathbf{X}_t + \mathbf{W}_{ho} * \mathbf{H}_{t-1} + \mathbf{W}_{co} \odot \mathbf{C}_t + \mathbf{b}_o) \\
\mathbf{H}_t &= \mathbf{O}_t \odot \tanh(\mathbf{C}_t)
\end{aligned}
\tag{2.10}
$$

where $*$ denotes the convolution operator and $\mathbf{F}_t$, $\mathbf{I}_t$, $\mathbf{O}_t$, $\mathbf{C}_t$, $\mathbf{H}_t$, $\mathbf{X}_t$ are respectively the forget gate, input gate, output gate, cell state, hidden state and input matrix. This architecture adds peephole connections that allow the network to look at the cell state to make decisions. This is highlighted in Figure 2.8.

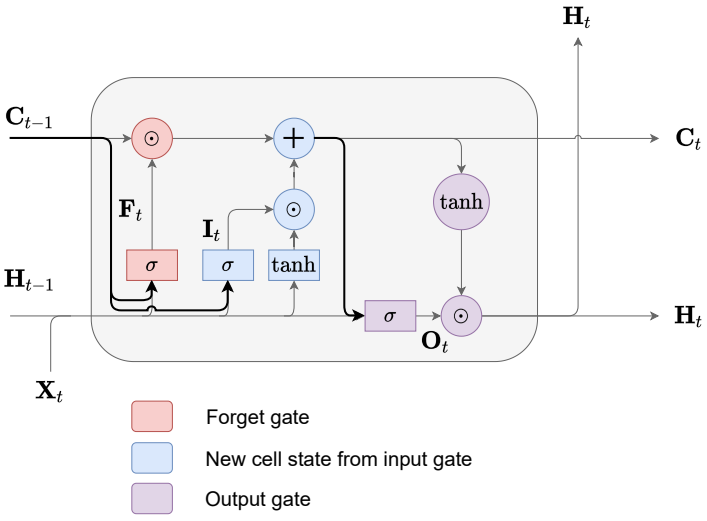


Figure 2.8 – Representation of a ConvLSTM network. New peephole connections, used to peep at the cell state, were highlighted with bold arrows.

RNNs facilitate the modeling of long-term dependencies through their potentially infinite memory. Comparatively, CNNs are limited to a fixed receptive field, which is cumbersome in case of sequences. However, the main drawback of the RNNs formulation is its limitation for

parallel computations. As the previous state needs to be computed to output the next one, it becomes mandatory to run the calculations sequentially. Yet, despite such limitation, RNNs have been used with great success for Natural Language Processing (Lipton [2015], Wu et al. [2016]), where the need for long-term dependencies outweighs the need for fast computation.

2.2 Object detection

In the previous section, we have detailed the base architecture of ANNs as well as specific types of layers. We now present the task of object detection, a common computer vision task.

Object detection aims at detecting instances of certain types of objects within images or videos. It has been an active field of research throughout the years due to its various practical applications (pedestrian detection, face recognition, tracking, etc.). Although the task of detection itself can be formulated in various ways, usually, in the literature, object detection refers to the prediction of rectangular boxes (bounding boxes) enclosing each object of interest. While the general usage of such formulation may be surprising as most existing objects are not of rectangular shape, its prevalence is probably due to the simplicity of annotating images with rectangular boxes, leading to the creation of multiple datasets (for instance Pascal VOC [Everingham et al., 2010] or MS-COCO [Lin et al., 2014] for general object detection, or BBD100K [Yu et al., 2020a] for vehicle detection).

Before the broad adoption of deep learning methods for image processing, object detection usually relied on a two steps pipeline: first, carefully selected discriminating features are extracted, then, the so-extracted features are used to detect and classify objects. Examples of such pipeline can be found with Haar-like features [Viola and Jones, 2001] for face detection or with Scale-Invariant Feature Transform (SIFT) descriptors [Lowe, 2004] for general object detection. More recently, thanks to the public release of the aforementioned large detection datasets as well as the astute leverage of feature extractors initially trained on classification tasks, deep learning based methods have completely supplanted older approaches for detection. Throughout the years, two main architectures have emerged: two-shot and one-shot detectors. The two-shot architecture was introduced with R-CNN [Girshick et al., 2014], which relied on non deep learning methods for some parts of its procedure. Further, R-CNN was improved to an end-to-end architecture with Fast R-CNN [Girshick, 2015] and Faster R-CNN [Ren et al., 2015]. Two-shot detectors are similar to the older detection methods: features are extracted in a first step and then used in a second one to provide with the final detection. However, due to the possibility to learn the whole pipeline, the two-shot detectors are much more accurate. The one-shot architecture was popularized by Liu et al. [2016] (SSD) and Redmon et al. [2016] (YOLO) in an effort to make the overall detection process faster. To do so, one-shot detectors process the images and predict the detections in one go, which avoids the computational overhead of two-shot detectors. However, due to the lack of object's features extraction, the gain in speed is achieved at the cost of a slight loss in accuracy.

This section aims at providing with an understanding of the overall object detection task. The rest of the section is divided as follows: we start by detailing the general formulation of the loss usually used, then, we review the two classical architectures (one-shot and two-shot detectors) and we finish by describing the usual metric used to evaluate detection tasks.

2.2.1 Classical object detection formulation and learning

The aim of object detection is to find an unknown number of objects within an image. An object can be formalized as the composition of a bounding box $b \in \mathcal{B}$ and an associated class $c \in \mathcal{C}$, where $\mathcal{B}$ and $\mathcal{C}$ are respectively the sets of all the possible boxes and classes (for clarity's sake we use the scalar notation b and c, rather than the vectorial one, to denote a box and a class). Therefore, the loss used to train the detection network is usually divided into two

parts. The first part, the *regression loss*, aims at evaluating if the predicted boxes correctly enclose the target objects. The second part, the *classification loss*, aims at evaluating the quality of the class prediction for each predicted box. Given an input image $\mathbf{X} \in \mathcal{X}$, we denote the associated target objects as $\mathbf{y} = \{(b_1, c_1), ..., (b_N, c_N)\} = \{\mathbf{b}, \mathbf{c}\}$ and the objects estimated by the network as $\mathcal{M}(\mathbf{X}) = \hat{\mathbf{y}} = \{(\hat{b}_1, \hat{c}_1), ..., (\hat{b}_M, \hat{c}_M)\} = \{\hat{\mathbf{b}}, \hat{\mathbf{c}}\}$ (M and N can be different). Let $\delta_{ij} = \{0, 1\}$ be an indicator of matching the i-th ground truth box to the j-th predicted box[2][3] (a box can be matched multiple times). We denote the indexes j that were matched positively to a ground truth box as Pos and the ones that were not matched as Neg. Then, the regression loss can then be formulated as:

$$\mathcal{L}_{reg}(\mathbf{b}, \hat{\mathbf{b}}) = \sum_{j \in Pos} \sum_{i} \delta_{ij} \left\| b_i - \hat{b}_j \right\|_b, \tag{2.11}$$

where $\|.\|_b$ is a distance between boxes. As it would make no sense to regress the negative examples that were not matched to a ground truth box, only the positive ones are accounted for. And the classification loss can be formulated as:

$$\mathcal{L}_{class}(\mathbf{c}, \hat{\mathbf{c}}) = \sum_{j \in Pos} \sum_{i} \delta_{ij} \mathcal{L}_c(c_i, \hat{c}_j) + \sum_{j \in Neg} \sum_{i} \delta_{ij} \mathcal{L}_c(c_i, \hat{c}_j), \tag{2.12}$$

where $\mathcal{L}_c(.,.)$ is a classification loss function. For the boxes that were negatively matched, a dummy class "background" is usually used to compute the loss. Grouping the two equations, the overall object detection loss can be formulated as:

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = \alpha \mathcal{L}_{reg}(\mathbf{b}, \hat{\mathbf{b}}) + \beta \mathcal{L}_{class}(\mathbf{c}, \hat{\mathbf{c}}), \tag{2.13}$$

where α and β are hyper-parameters setting the relative importance of each part of the loss. Depending on the type of detection network used, parts of the loss can be impacted differently. For instance, one-shot detectors tend to generate a lot of negative boxes, which negatively impact the classification loss as the sum on the set Neg can overwhelm the classification loss (Equation 2.12).

2.2.2 One-shot vs Two-shot detection architectures

We have detailed the classical loss used to train detection networks, we now review the typical existing architectures. Although object detection has been widely studied over the years, most of the deep learning models that have been proposed boil down to two main types of architectures: one-shot and two-shot detectors.

The two-shot detectors process the images in a two steps pipeline. First a backbone network extracts a meaningful representation from images. Then, this extracted representation is processed by a Region Proposal Network (RPN) to predict a set of possible objects (Region of Interest [RoI]). After which, these RoIs are extracted from the feature map, usually using a RoI pooling layer. Finally, all the extracted regions are regressed and classified to obtain the predicted objects. The overall two steps detection pipeline is shown in Figure 2.9. Using such an approach has the main advantage of explicitly extracting the features related to a possible object. This allows a more focused processing of features of interest and, possibly, the combination of multiple tasks (for instance detection and segmentation He et al. [2017]). However, the downside is the low detection speed. As all the RoIs are processed one by one, this make for a suboptimal computation process.

Opposite to the two-shot detectors are the one-shot detectors. They are very similar in spirit with the RPN branch of the two-shot detectors. However, rather than using the predicted boxes to extract features for a second processing steps, the class is directly associated

[2]The matching strategy is highly dependent on the proposed architecture and will therefore not be described here.

[3]It is to be noted that few authors (Law and Deng [2018], Duan et al. [2019]) have based their architecture on predicting points of interests rather than boxes. In such case the points are matched instead of the boxes.

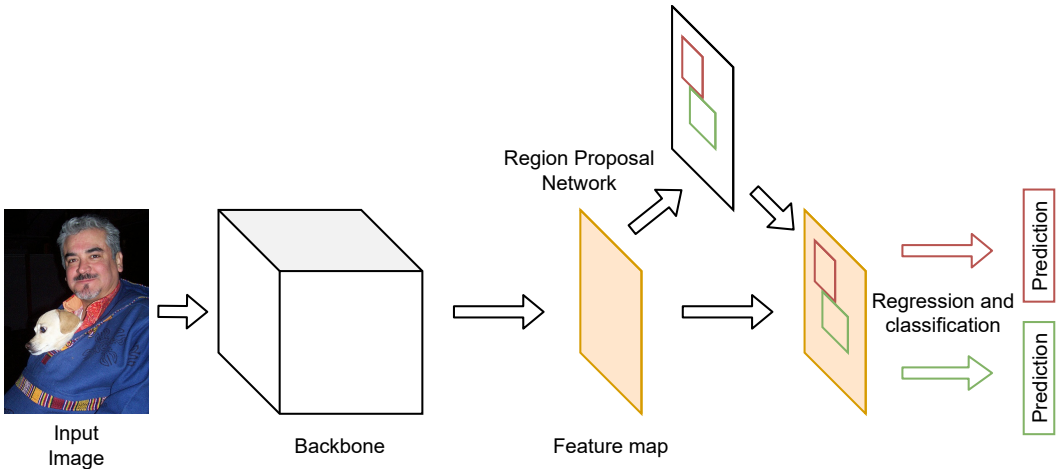


Figure 2.9 – Detail of the two-shot detectors architecture. A feature map is first computed from the image. Using this representation, Regions of Interest (RoIs) are predicted and, finally, the RoI are extracted from the feature map and used for prediction.

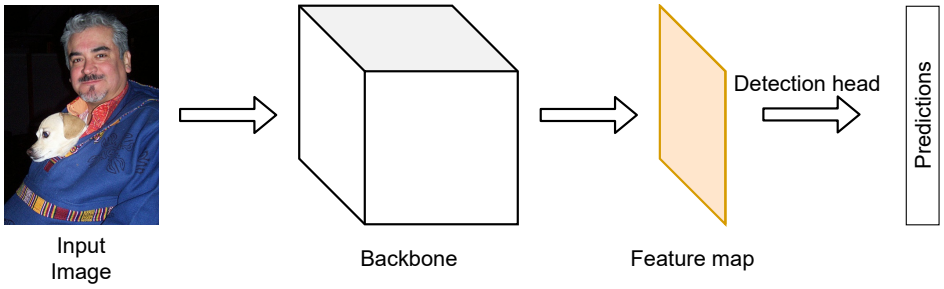


Figure 2.10 – Illustration of the one-shot detectors architecture. All the boxes are directly predicted from the feature map, making for a faster processing.

to the predicted boxes. Obviously due to the avoidance of the extraction step, these networks are order of magnitude faster than the two-shot detectors. This increase in speed, however, usually comes at the cost of a slight loss in accuracy when compared with two shots detectors. The functioning of one shot detectors is illustrated in Figure 2.10.

While using different pipelines, both architectures are likely to detect a given object multiple times due to the large number of proposed boxes (c.f Figure 2.11). In order to filter boxes that correspond to the same object, Non-Maximum Suppression (NMS) is usually used. The NMS algorithm is illustrated in Figure 2.11 and works as detailed hereafter. First, a detection network outputs a list of predicted boxes $\hat{\mathbf{y}}$ with a list of associated confidence score (i.e a score of the confidence that the network has of a box as being an actual detection, usually the classification accuracy is used). Then the following steps are repeated until no box is left in $\hat{\mathbf{y}}$:

- the box with the highest confidence score is selected from $\hat{\mathbf{y}}$,
- this box is compared with all the remaining boxes in $\hat{\mathbf{y}}$,
- if a remaining box is too close to a box from $\hat{\mathbf{y}}$, then this box is considered a duplicate detection and therefore removed from $\hat{\mathbf{y}}$
- the selected box is added to the list of selected boxes.

Usually the distance used to measure if two boxes are close or not is the Intersection over Union (IoU) (see Figure 2.12). It is to be noted that variations of the NMS algorithm have been proposed such as the Soft-NMS [Bodla et al., 2017] (which changes the policy used to remove a box from $\hat{\mathbf{y}}$), however, the underlying algorithmic steps are usually unchanged.

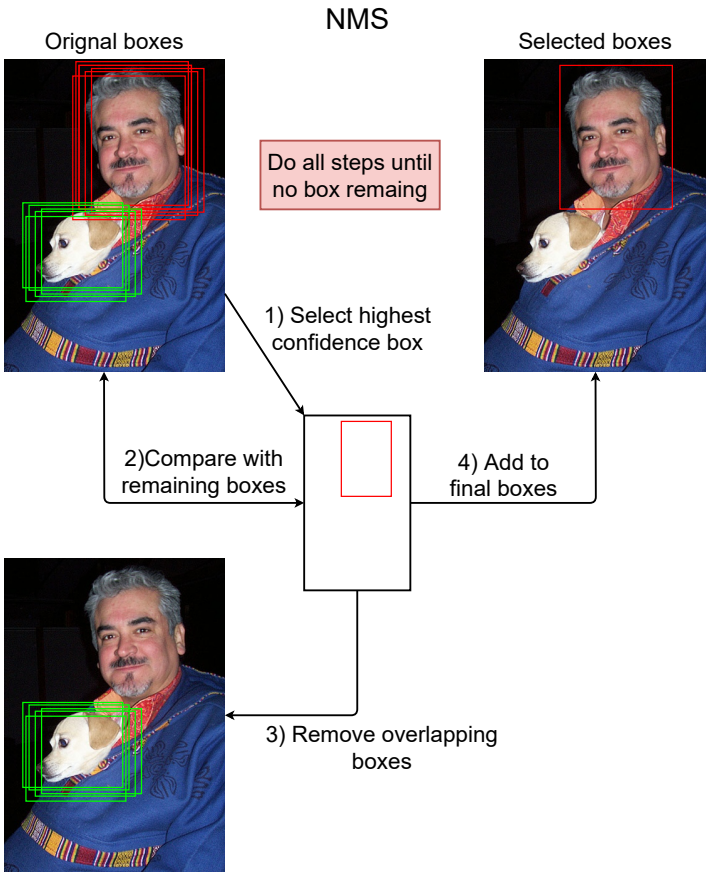


Figure 2.11 – Details of the Non-Maximum Suppression algorithm.

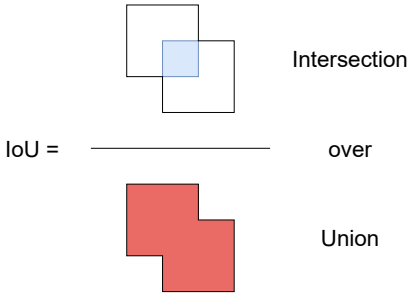


Figure 2.12 – Illustration of the Intersection over Union.

2.2.3 Evaluation: mean Average Precision

So far we have seen how detection networks are trained and what are the two main architectures in use. We now detail the evaluation metric used for most of the detection tasks: the mean Average Precision (mAP).

The mAP is a metric that expresses how good a network is at predicting the correct boxes while avoiding false positive detections. It is based on the computation of the precision and the recall, which are respectively defined as the number of true positive elements over the total of positive elements and the number of true positive elements over the total of elements to be predicted (c.f Figure 2.13). The mAP is then defined as the mean value of the discrete Average Precision (AP) of each class to be detected, where the discrete AP is defined as:

$$AP = \frac{1}{N} \sum_{r \in \{0, 0+\frac{1}{N-1}, \ldots, 1\}} p_{interp}(r), \tag{2.14}$$

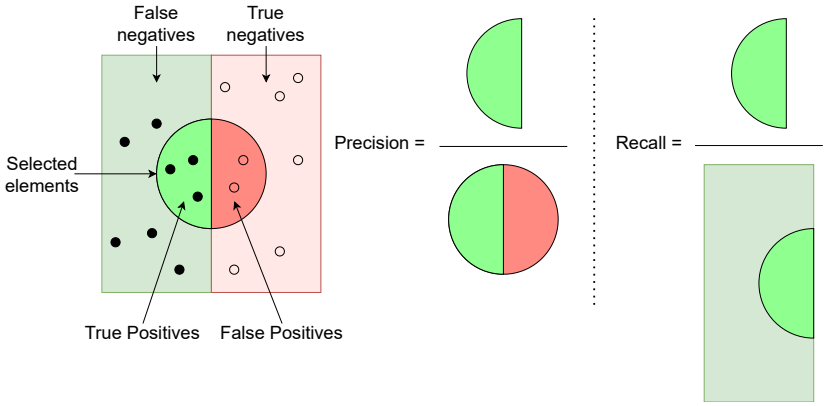


Figure 2.13 – Visual representation of the precision and recall values. Precision is number of true positive elements over the total of positive elements. Recall is the number of true positive elements over the total elements to be predicted. Figure adapted from wikipedia.

where r is the recall value and

$$p_{interp}(r) = \max_{\tilde{r}:\tilde{r}\geq r} p(\tilde{r}), \tag{2.15}$$

with p being the precision function. The computation of each value of Equation 2.15 is visually represented in Figure 2.14.

Total boxes to be detected: 5

Rank	Is True Positive ?	Precision	Recall
1	True	1	0.2
2	True	1	0.4
3	False	0.67	0.4
4	True	0.75	0.6
5	True	0.8	0.8
6	False	0.67	0.8
7	False	0.57	0.8
8	False	0.5	0.8
9	False	0.44	0.8
10	True	0.5	1

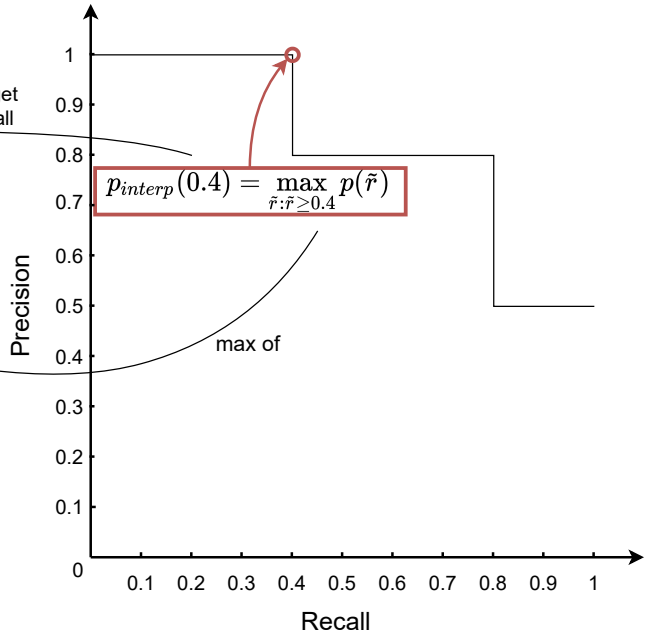


Figure 2.14 – Visual representation of the mAP formula.

2.3 Connectionist Temporal Classification

In the previous section, we have detail how networks are trained for object detection when the position of the objects is known within input data. Such training is usually done with a combined regression and classification loss. However, this two-part loss is conditioned on the matching of the network propositions with the ground truth data. This is not possible when the position of the objects is not known beforehand. For such a case, the Connectionist

Temporal Classification (CTC) loss [Graves et al., 2006] was introduced. Using this loss, it becomes possible to train a neural network on temporally unsegmented data.

The aim is, given the input sequence $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_T)$, where $\mathbf{x} \in \mathcal{X}$ the input space, to predict the correct target sequence $\mathbf{z} = (\mathbf{z}_1, \ldots, \mathbf{z}_U)$, where $\mathbf{z} \in \mathcal{Z}$ the target space (T and U respectively represent the size of the input/target sequences). $\mathcal{Z}$ is defined as $\mathcal{Z} = L*$ where $L*$ is the set of all possible sequences over the finite alphabet L. In the rest of the thesis, we call an element $\mathbf{l}$ of $\mathcal{Z}$ a *labelling*, $\mathbf{z}$ being the *target labelling*. In the setting we are interested in, $U \leq T$ and therefore the two sequences cannot be *a priori* aligned. Such set-up can arise for problem such as the optical recognition of word images (see Figure 2.15). In this problem, series of slices that may or may not contain letters need to be classified in order to output the target sequence, which in the case of the example is the word "too".

In order to ease the comprehension of the CTC loss, we start by detailing the output representation that allows a neural network to be trained with such a loss. Then, building up on this representation, we dive in the details of the CTC loss.

2.3.1 From network output to labelling

Let's proceed with the toy problem of optical recognition of word images (without any language model)[4]. Suppose that the image from which to predict a word is provided as a time series of pixels slices as shown in Figure 2.15. We aim, given a network ("CTC network" hereafter) that predicts the probability of each letter of the alphabet L to be contained in each slice, to obtain the most probable labelling $\mathbf{l} \in \mathcal{Z}$.

Intuitively, one would consider selecting the most likely letter at each time step and then convert the resulting sequence into a labelling. Such conversion is done by introducing a mapping $\mathcal{B}$ that collapses a series of identical letters into one letter as illustrated in Figure 2.15. For this mapping to be fully functional one needs to introduce a *blank* symbol ('-') to serve as separator. Indeed, without such symbol, labellings with series of identical letters such as 'too' can not be obtained through $\mathcal{B}$ as they would be collapsed to 'to'. We denote the alphabet augmented with the blank symbol as $L' = L \cup \{blank\}$.

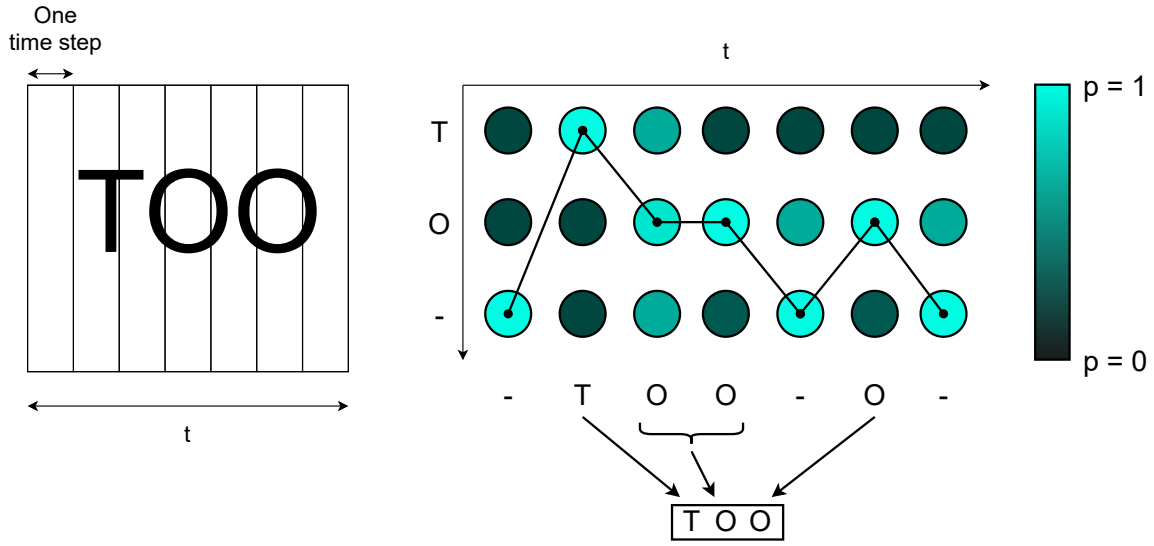


Figure 2.15 – Example of series of text input. Some of the inputs may have multiple labels, which limits the meaning of the segmentation.

Using the most probable prediction at each time step to generate the labelling is called "best path decoding". It consists in considering that the argmax of the prediction at each time step will form the best prediction path. However, given the mapping function, several paths

[4]This peculiar example and Figure 2.15 are based on this post.

may lead to the desired labelling (for instance $\mathcal{B}(-TOO-O-) = \mathcal{B}(-T-O-OO) = TOO$). Therefore, the probability of a labelling $\mathbf{l}$ given the input sequence $\mathbf{x}$ is actually computed as:

$$p(\mathbf{l}|\mathbf{x}) = \sum_{\pi \in \mathcal{B}^{-1}(\mathbf{l})} p(\pi|\mathbf{x}) \tag{2.16}$$

where π is a path and $\mathcal{B}^{-1}(\mathbf{l})$ is the set of all paths π leading to $\mathbf{l}$. The probability of a given path is expressed as:

$$p(\pi|\mathbf{x}) = \sum_{t=1}^{T} y_{\pi_t}^t, \forall \pi \in L'^T, \tag{2.17}$$

where $y_{\pi_t}^t$ is the computed probability at time step t of the path's symbol at time step t and $0 \leq p(\pi|\mathbf{x}) \leq 1$. Because computing the probability of $\mathbf{l}$ implies that each path should be tested, there is no tractable algorithm to compute it. However, Graves et al. [2006] propose a method based on a modification of the forward-backward algorithm, used to compute the CTC loss called "prefix-search decoding", that can be used to *approximate* the probability of a given labelling.

2.3.2 Training a CTC network: loss and dynamic programming

In the previous subsection we have seen how to use a CTC network to predict labellings of variating size from input sequences. We now consider the loss proposed to train such a network. To train a CTC network $\mathcal{M}$, one considers minimizing the negative log likelihood:

$$O(\mathcal{M}) = -\sum_{(\mathbf{x},\mathbf{z})\sim S} \ln(p(\mathbf{z}|\mathbf{x})), \tag{2.18}$$

where S is the distribution from which the training data is drawn, $\mathbf{x}$ and $\mathbf{z}$ are respectively input and target sequences. As the CTC loss is based on temporal data, we aim to compute the following derivative to be able to apply the backpropagation algorithm:

$$\frac{\partial O((\mathbf{x},\mathbf{z}),\mathcal{M})}{\partial y_k^t} = -\frac{\partial \ln(p(\mathbf{z}|\mathbf{x}))}{\partial y_k^t}, \tag{2.19}$$

where y_k^t the output unit k at time t of the prediction network. However, Equation 2.16 suggests that to compute $p(\mathbf{z}|\mathbf{x})$, all the paths that can be mapped through $\mathcal{B}$ to the target labelling $\mathbf{z}$ should be evaluated, which is not tractable.

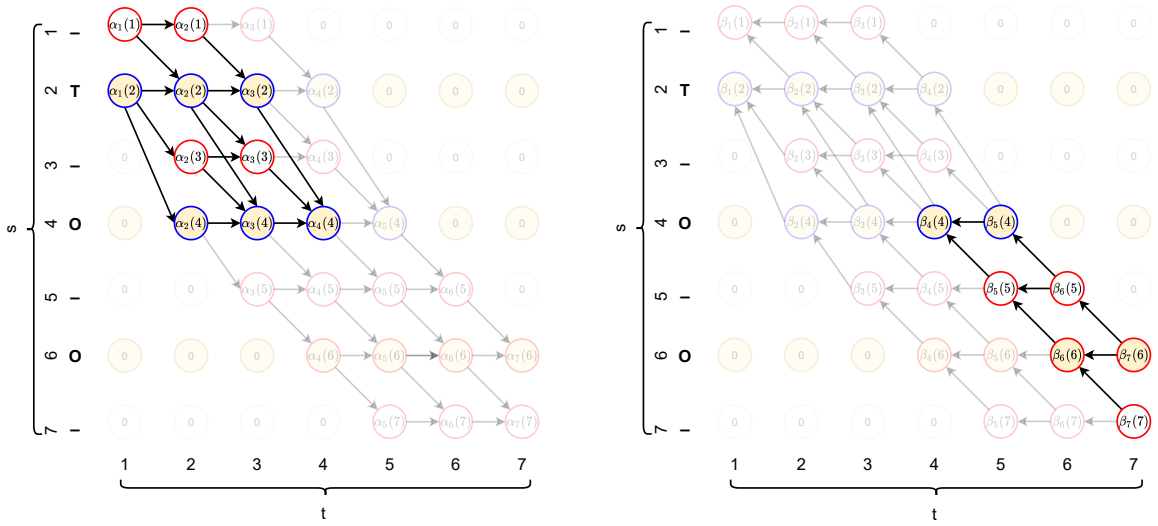
To circumvent this problem, Graves et al. [2006] propose to use a forward-backward algorithm based on dynamic programming. The main idea is to use two variables, the forward one, that computes the probability of being in state s at time t given the beginning of the sequence from 0 to t:

$$\alpha_t(s) \overset{\text{def}}{=} \sum_{\substack{\pi \in \Pi^T \\ \mathcal{B}(\pi_{1:t}) = \mathbf{l}_{1:s}}} \prod_{t'=1}^{t} y_{\pi_{t'}}^{t'}, \tag{2.20}$$

where $\pi_{i:j}$ represents the path from time i to time j, $\mathbf{l}_{i:j}$ represents the labelling from index i to j and Π^T is the set of all possible paths of length T. And the backward one, that computes the probability of being in state s at time t given the end of the sequence from t to T:

$$\beta_t(s) \overset{\text{def}}{=} \sum_{\substack{\pi \in \Pi^T \\ \mathcal{B}(\pi_{t:T}) = \mathbf{l}_{s:|\mathbf{l}|}}} \prod_{t'=t}^{T} y_{\pi_{t'}}^{t'}. \tag{2.21}$$

By multiplying them, one may compute the probability γ of all the paths going through $\mathbf{l}_s$ at time t and, therefore, by summing over all s one get $p(\mathbf{l}|\mathbf{x})$. This is illustrated in Figure 2.16. It is to be noted that in order to allow for blanks in the path, rather than using $\mathbf{l}$, the



(a) Details of the forward variable α. All the paths leading to $\alpha_4(4)$ are highlighted.

(b) Details of the backward variable β. All the paths leading to $\beta_4(4)$ are highlighted.

Figure 2.16 – Illustration of the forward and backward variables applied to the example "too". The probability of the letter "o" at time $t = 4$ can be obtained by multiplying $\alpha_4(4)$ and $\beta_4(4)$.

label sequence $\mathbf{l}'$ is used. This sequence has blanks symbols inserted in-between the original symbols as well as at the beginning and the end of $\mathbf{l}$ and is therefore of size $2|\mathbf{l}| + 1$.

As we aim to compute $p(\mathbf{z}|\mathbf{x})$, the first step is to notice that the probability of a given path going through a specific label can be obtained from α and β as follows:

$$\begin{aligned}
\alpha_t(s)\beta_t(s) &= \sum_{\substack{\pi \in \mathcal{B}^{-1}(\mathbf{l}): \\ \pi_t = \mathbf{l}'_s}} y^t_{\mathbf{l}'_s} \prod_{t=1}^{T} y^t_{\pi_t} \\
\frac{\alpha_t(s)\beta_t(s)}{y^t_{\mathbf{l}'_s}} &= \sum_{\substack{\pi \in \mathcal{B}^{-1}(\mathbf{l}): \\ \pi_t = \mathbf{l}'_s}} \prod_{t=1}^{T} y^t_{\pi_t} \\
\frac{\alpha_t(s)\beta_t(s)}{y^t_{\mathbf{l}'_s}} &= \sum_{\substack{\pi \in \mathcal{B}^{-1}(\mathbf{l}): \\ \pi_t = \mathbf{l}'_s}} p(\pi|\mathbf{x}).
\end{aligned} \tag{2.22}$$

Looking at Figure 2.16, it is obvious that, to compute $p(\mathbf{l}|\mathbf{x})$, one simply have to sum the probabilities of each path going through each label at any time t. Hence, we have:

$$p(\mathbf{l}|\mathbf{x}) = \sum_{s=1}^{|\mathbf{l}'|} \frac{\alpha_t(s)\beta_t(s)}{y^t_{\mathbf{l}'_s}}. \tag{2.23}$$

Therefore, replacing $\mathbf{l}$ by $\mathbf{z}$ the target labelling, we have:

$$\frac{\partial p(\mathbf{z}|\mathbf{x})}{\partial y^t_k} = \frac{1}{y^{t^2}_k} \sum_{s \in lab(\mathbf{z},k)} \alpha_t(s)\beta_t(s), \tag{2.24}$$

where $lab(\mathbf{z}, k) = \{s : \mathbf{l}'_s = k\}$ is the set of s where output label k occurs (i.e for the example word "too", $lab(\mathbf{z}, \text{'o'}) = \{4, 6\}$). Finally, starting from Equation 2.19 the derivative can be computed as follows:

$$-\frac{\partial \ln(p(\mathbf{z}|\mathbf{x}))}{\partial y^t_k} = \frac{1}{p(\mathbf{z}|\mathbf{x})} \frac{\partial p(\mathbf{z}|\mathbf{x})}{\partial y^t_k}. \tag{2.25}$$

This clever computation proposed by Graves et al. [2006] paved the way for the application of the deep learning framework on tasks that previously seemed out of reach for such training

paradigm. In particular, speech and handwriting recognition, where segmentation labelling is difficult to obtain, have largely benefited from the CTC loss (Graves et al. [2013], Graves and Jaitly [2014], Chorowski et al. [2015], Amodei et al. [2016], Greff et al. [2017]).

2.4 Conclusion

Deep learning is a machine learning method that aims to learn in an end-to-end manner a prediction model given a task. Although the base formulation of neurons can be traced back to the forties [Mcculloch and Pitts, 1943], it is only in the recent years with combination of backpropagation [Kelley, 1960], increase in computation power and the sharing of huge datasets (Russakovsky et al. [2015], Lin et al. [2014]), that deep learning was able to take off.

Notably, the task of object detection has largely benefited from the deep learning's rise and has seen impressive improvements lately. Throughout the years, two main architectures have emerged, namely one-shot and two-shot detectors. One-shot detectors were popularized by Liu et al. [2016] and Redmon et al. [2016]. They are fast and usually aimed towards real-time detection, however, they fall behind the two-shot detectors when it comes to accuracy. The two-shot architecture was initiated and improved through the series of R-CNN papers (Girshick et al. [2014], Girshick [2015], Ren et al. [2015]). Opposite to the one-shot detectors, they are usually slow but accurate. This is due to the two-step pipeline used, that allows for a clear extraction of each object features. Such extraction can be used for a more focused or multitask processing such as carried by He et al. [2017], doing both instance detection and instance segmentation. While these two types of architecture differ in their processing, they both rely on a dual loss, based on classification and regression. This loss, although effective, requires the position of the objects to be known, which is a strong *a priori*.

For cases when positioning is not known, the CTC loss [Graves et al., 2006] was introduced. The CTC loss relies on dynamic programming to compute the probability of labels from all the possible output paths. Using this formulation, it becomes possible to estimate the position of objects. However, in its current formulation, the CTC loss is limited to one dimension, meaning that there should theoretically be only at most one object per step in the target dimension.

Part II

Contributions

Chapter 3

Object detection in Compressed JPEG images

“ Time flies like an arrow; fruit flies like a banana ”

Robin Hood

Contents

This chapter is based upon our contributions Deguerre et al. [2019] and Deguerre et al. [2021].

Deep architectures, especially Convolutional Neural Networks (CNN) have become a standard for object detection [Zou et al., 2019b]. Most of the proposed architectures rely on the same two components: a feature extractor (backbone) pre-trained on a classification task and a detection head to output the box predictions. Impressive results were achieved [Liu et al., 2021] on datasets growing in complexity such as Pascal VOC [Everingham et al., 2010] or MS-COCO [Lin et al., 2014]. However, even though most of the images are compressed in order to limit the storage and the transfer bandwidth requirements, state-of-the-art detection architectures are designed for processing RGB inputs. Hence the usual procedure for any detection task follows these steps:

- The image is uncompressed and possibly pre-processed, namely with normalization and/or resizing operations.
- Then the detection task is performed, typically using a deep network.

Although effective, this procedure has some drawbacks as the image decoding step induces a computational cost. Moreover, the involved deep architectures may include a large amount of parameters, due to the RGB image resolution, making the network computationally expensive. These facts hinder the large scale deployment of detection networks for applications with real-time constraints such as city surveillance [Ide et al., 2016] or road traffic monitoring and management [Wang et al., 2017].

In this thesis we are concerned with the surveillance of road tunnels in Paris and, in particular, with the Automatic Incident Detection (AID) system. The AID system has been made mandatory by law (annex 2 of circular note 2000-63 of 25 August 2000) in order to improve incident detection, monitoring and analysis. The main events that are to be detected have been listed by the CEnter for TUnnel studies (CETU) and, in this work, we choose to focus on the detection of vehicles stopped on unauthorized area (i.e hard shoulders, zebra stripes, see Figure 3.1). It is an important part of the surveillance task as it allows the operators to be quickly notified of potential risky situations within the tunnels. At the moment, the detection of vehicles in Paris' tunnels is done using classical image processing. While functional, this approach relies on heuristics that require a manual calibration. To remove this limitation, Actemium seeks to upgrade the existing detection system with newer deep learning based methods. Directly applying existing deep architectures to this detection problem would reveal costly as this task needs to run in real time (up to 25 Frames Per Second (FPS) at full temporal resolution) on a high number of cameras (about 2000). Therefore, in this chapter, we aim to skip the image decompression step and to perform object detection on compressed image representation to reduce the computation and memory footprint.

Tunnel's cameras have two operating modes: they can either output a MPEG4 part-2 coded stream or JPEG compressed snapshots on demand. Ideally, we would like to run object detection on the MPEG4 part-2 coded streams. However, detection on compressed videos raises two main issues: first, how to run detection on compressed frames and second, how to take advantage of the encoded temporal information. In order to simplify the addressed problem, we choose to first focus on object detection in compressed JPEG images, removing the temporal aspect. As the JPEG and MPEG4 part-2 compressions share processing steps, it is likely that a method developed for JPEG compressed images will be applicable to MPEG4 part-2 videos. Although previous work has demonstrated the possibility to carry *classification* in the compressed JPEG domain [Gueguen et al., 2018], *object detection* brings new challenges. Indeed, as the JPEG compression transforms images into a tiled frequency space (8 pixels upper bound) through Discrete Cosine Transform (DCT), it raises the question whether a detection network is able to efficiently map the frequency domain into a spatial domain in order to output the position of the objects in the original image. Futhermore,

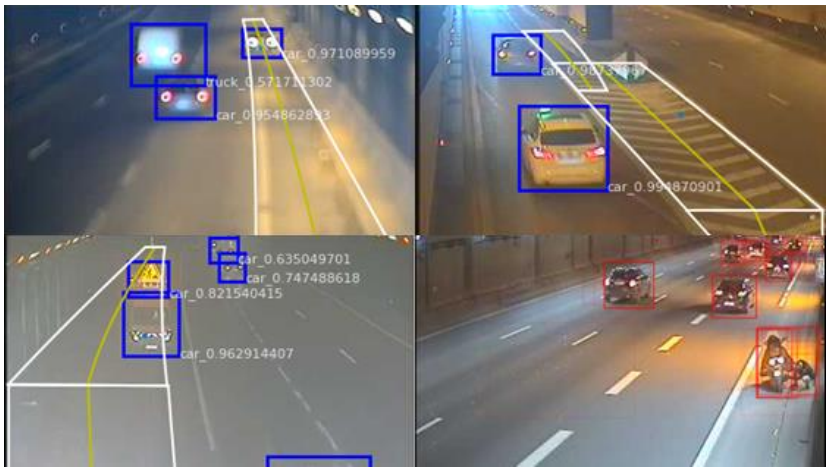


Figure 3.1 – Examples of unauthorized areas (white shapes) and detected vehicles on the road. Images from Actemium's website.

detection in a JPEG compressed domain is faced with the chrominance (C_r and C_b components) sub-sampling that leads to a resolution change when compared with the luminance component (Y component), see Chapter 1, subsection 1.1.3.

In this chapter, not only do we demonstrate that object detection in the compressed domain is achievable, but also that very close detection performances to those of RGB-based architectures can be reached with a significant speed up gain. This is achieved by leveraging the block compression of JPEG images to provide the detection networks with object location information for the prediction while using inputs in the frequency domain. Additionally, we investigate the use of the sole Y channel as input of the proposed detection networks so as to get rid of the resolution change problem. Based on this solution, we empirically demonstrate that using only the Y channel is enough to reach an accuracy equivalent to the one of networks relying on YC_bC_r.

The remainder of the chapter is divided as follow: first, we review existing methods for object detection, as well as for the processing of compressed data, and show their limitation for the task at hand. Then, we present the proposed method to run object detection on compressed JPEG images and, finally, we discuss the results obtained on the various test datasets.

3.1 Detecting objects in images

Object detection aims at detecting all the objects of interest within a given image. As detailed in section 2.2, the object detection task is usually cast as the prediction of bounding boxes rather than the prediction of the objects themselves. Most of the existing deep neural networks for detection rely on pre-trained classification modules to help improve the final accuracy. For instance, detectors such as Fast R-CNN [Girshick, 2015], Faster R-CNN [Ren et al., 2015], SSD [Liu et al., 2016] or FSSD [Li and Zhou, 2017] use the VGG16 network [Simonyan and Zisserman, 2015] as a backbone, while R-FCN [Dai et al., 2016b] is instead based on deep residual network [He et al., 2016]. In general, detection networks are not linked to a specific classification module: they can easily be used with other classification backbones (provided the dimensions of input images and network outputs are adjusted accordingly). As the majority of the available backbones were trained on RGB images, most of the detection methods use RGB images as input. Still, Gueguen et al. [2018] proved that at least classification is feasible using JPEG compressed images, hinting towards a possible adaptation to the object detection tasks.

In order to get a better understanding of both object detection and the use of compressed

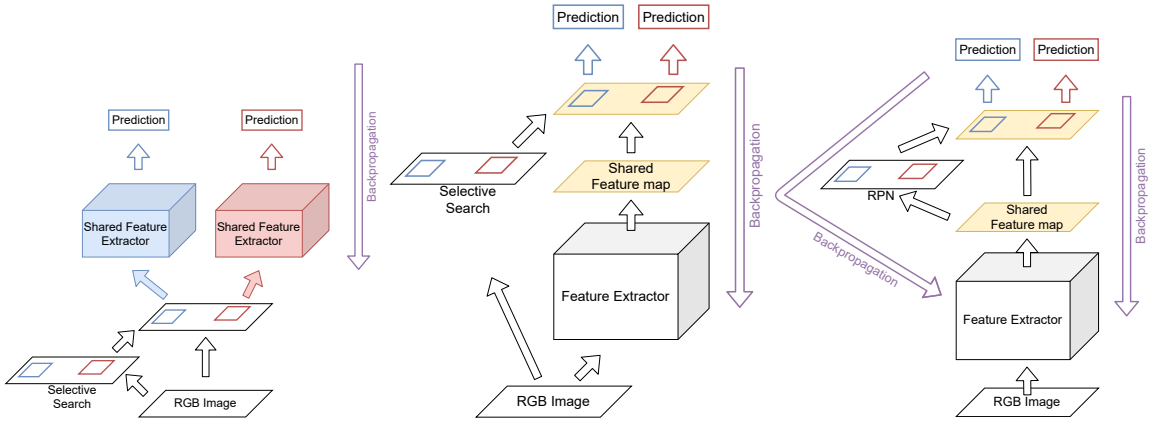
JPEG image as inputs, we first details the existing related works for object detection in RGB images. Then, we review the literature about the usage of the compressed representation for image and video processing.

3.1.1 Object detection on RGB images

Methods for object detection are broadly based on RBG inputs and rely on two main deep architectures: two-stage detectors and one-stage detectors. The former networks are based on a region proposal stage followed by a box classification and/or regression stage while the latter architectures predict a set of pre-defined boxes as object or background. While based on a different architecture, both networks use a similar joint classification and regression loss. Therefore, except for specific cases, we do not detail the loss function. All the methods reviewed hereafter are summarized in Figure B.1 (see Appendix B) to help better grasp the evolution of the detection architectures throughout the years, and, Table 3.1 regroups all their performances.

Two-shot detectors

Two-stage detectors (c.f Figure 2.9) were introduced with R-CNN [Girshick et al., 2014]. This first approach uses Selective Search [Uijlings et al., 2013] to extract object proposals that are then refined through a classification network to remove false positives. This approach, while effective, raises multiple issues. As classification networks expect a fixed size input, the proposals need to be resized before the classification step, potentially deteriorating accuracy. Furthermore, because of the Selective Search, the whole network can not be trained in an end-to-end fashion. The first issue was addressed through Spatial Pyramid Pooling (SPP) [He et al., 2014]. To avoid the loss in accuracy due to the wrapping, SPP introduces a new pooling layer that adapts to any size of input and generates a fixed size representation. This SPP layer was further improved in Fast R-CNN [Girshick, 2015]. Fast R-CNN replaces the SPP layer with a simpler Region of Interest [RoI] pooling layer to provide with complete backpropagation on the main network, from the classification head, down to the feature extractor. Yet, this architecture still lacks the possibility to be completely trained end-to-end, as the object proposals were still extracted using Selective Search. Faster R-CNN [Ren et al., 2015] solves this issue by replacing the Selective Search algorithm with a Region Proposal Network (RPN). These improvements lead to the possibility to train each part of the detection pipeline, thus leading to the first "true" two-shot deep detection network (the evolution of the architecture is shown in Figure 3.2).



(a) R-CNN (b) Fast R-CNN (c) Faster R-CNN

Figure 3.2 – Evolution of the R-CNN architectures, from the R-CNN (a) to Faster R-CNN (c). Fast R-CNN (b) improves on R-CNN mostly through the shared feature map. Faster R-CNN adds a RPN network, allowing to use backpropagation on both object proposal and classification.

As shown in Figure B.1 Faster-RCNN architecture serves as foundation for a large part of the two-shot detectors: Multi-task Network Cascades (MNC) [Dai et al., 2016a], R-FCN [Dai et al., 2016b], CoupleNet [Zhu et al., 2017b], Mask R-CNN [He et al., 2017], Feature Pyramid Network (FPN) [Lin et al., 2017a], Cascade R-CNN, [Cai and Vasconcelos, 2018], InterTwiner [Li et al., 2019a], TridentNet [Li et al., 2019b], Corner Proposal Network (CPN) [Duan et al., 2020]. However, while all these methods were built on the same base architecture, different types of approaches can be distinguished. For instance, MNC [Dai et al., 2016a] and Mask R-CNN [He et al., 2017] use a multitask loss to train their network simultaneously on both object detection and segmentation. Region-based Fully Convolutional Network (R-FCN) [Dai et al., 2016b] and CoupleNet [Zhu et al., 2017b] remove the classification head applied to each object proposal by relying on k^2 position-sensitive score maps. As each of the score map contains the class probability relative to the $k \times k$ RoIs pooled from the object proposals, classification can be directly obtained for each proposal. Taking advantage of the RoI pooling layer, InterTwiner [Li et al., 2019a] proposed to use optimal transport to reduce the discrepancy between the embedded representation of similar objects across various images. TridentNet [Li et al., 2019b] uses dilated convolutions to capture information at multiple scales. Cascade R-CNN [Cai and Vasconcelos, 2018], adds a cascaded detection pipeline using increasing Intersection over Union (IoU) thresholds to select the positive examples used for training. And, Corner Proposal Networks [Duan et al., 2020], replace the anchor boxes-based RPN with an anchor free detection that detects top-left and bottom-right corner of each box to get the object proposals. Lastly, it is important to take note of the Feature Pyramid Networks (FPN) [Lin et al., 2017a], as they are largely used throughout the literature and especially for one-shot detectors. FPNs are based on a multi-scale pyramidal prediction and add an up-sampling pyramid to the original feature extractor in order to provide each scale with information from lower scales.

Apart from the Faster R-CNN based methods, few have proposed improvements that are not directly linked to any network. SNIPER [Singh et al., 2018] extract chips of interest to compute the feature extraction step only on targeted parts of large images. CBNet [Liu et al., 2020a] stacks multiple backbones to improve the accuracy of the networks. And, finally, Bag Of Freebies [Zhang et al., 2019] proposes with a series of methods to improve detection accuracy of any type of network.

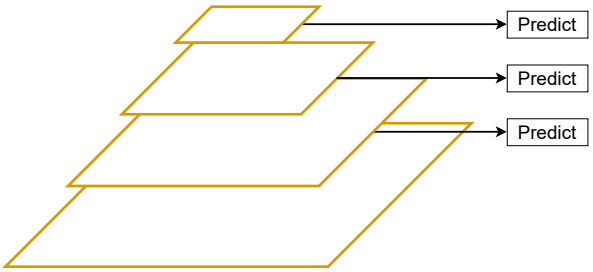
As of today, CPNs [Duan et al., 2020] represent the best two-shot detection networks for a balance between accuracy and speed of detection (39.7 mAP at 43.3 FPS on MS-COCO, c.f Table 3.1). However it is worth noticing that such speed was obtained using an high end NVIDIA V100 GPU worth between 9000-10000 euros, which would make large scale deployment order of magnitude too costly (this holds true even if we consider replacing the V100s with a high end "low cost" GPUs). Furthermore older architectures such as Faster R-CNN [Ren et al., 2015] or FPN [Lin et al., 2017a] are too slow to be usable for realtime detection even on low cost GPUs. Overall, speed and resources requirements make the two-shot detectors impractical for real-time applications, especially at large scale.

One-shot detectors

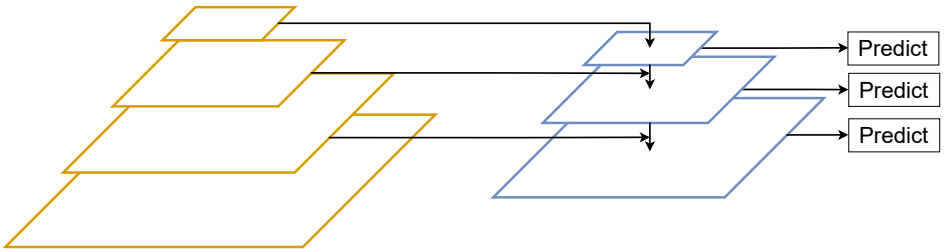
One-shot detectors (c.f Figure 2.10) are based on the prediction of densely pre-set bounding boxes. They were popularized by SSD [Liu et al., 2016] and YOLOs [Redmon et al., 2016, Redmon and Farhadi, 2017] architectures. Throughout the years, three main types of architectures have emerged: FPN-based, prior-box free and transformer-based.

FPN-based networks are the most similar to the original YOLO/SSD formulation and make up a major proportion of the existing literature (Deconvolutional SSD (DSSD) [Fu et al., 2017], RetinaNet Lin et al. [2017b], RefineDet [Zhang et al., 2018], Neural Architecture Search FPN (NAS-FPN) [Ghiasi et al., 2019], SpineNet [Du et al., 2020] YOLOv3 Redmon and Farhadi [2018], YOLOv4 [Bochkovskiy et al., 2020], Scaled YOLOv4 [Wang et al., 2021]). FPNs [Lin et al., 2017a] were originally designed for two-shot detectors, however they provide

with a nice synergy with one-shot detectors. Indeed, the added reversed pyramid allows to improve the detection by providing each detection layer at a given scale with information from lower scales (see Figure 3.3).



(a) Classical pyramidal feature hierarchy.



(b) Feature Pyramid Network architecture.

Figure 3.3 – The classical feature pyramidal architecture (a) and its FPN counterpart (b). As the information can flow back to higher level in the FPN architecture, it allows for an improved accuracy. Illustration modified from this post from "Towards data science".

FPN-based networks mostly improve on limitations from the original SSD and YOLO architectures. For instance, RetinaNet [Lin et al., 2017b] proposed the focal loss to alleviate the background imbalance problem of one-shot detectors. The original YOLOs were improved multiple times with YOLOv3 [Redmon and Farhadi, 2018], YOLOv4 [Bochkovskiy et al., 2020] and Scaled-YOLOv4 [Wang et al., 2021], mostly by applying various improvements, such as specific network scaling rules [Tan and Le, 2019] or training freebies [Zhang et al., 2019]. Scaled-YOLOv4 is the most impressive of all the YOLOs with a speed of 1774 FPS[1] while maintaining an accuracy of 22.0 (MS-COCO), similar to the YOLOv2 architecture. Taking another approach, others have also tried to improve the FPN module itself. DSSD [Fu et al., 2017] uses deconvolution layers to learn the transition between scales in the ascending pyramid. RefineDet [Zhang et al., 2018] adds a third series of layers after the FPN to better refine the detection of objects. Based on [Tan and Le, 2019], EfficientDet [Tan et al., 2019], proposes a new detection head, BiFPN, that stacks multiple FPN layers. Finally, a small part of the FPN-networks were learned using the Neural Architecture Search (NAS) [Zoph and Le, 2017] framework (NAS-FPN Ghiasi et al. [2019], SpineNet Du et al. [2020], MobileNetv3 Howard et al. [2019]). The main difference between the proposed methods being the search space used during the NAS.

Moving away from the FPN framework, another series of papers based their architectures on the anchor-free detectors. This type of network was introduced by CornerNet [Law and Deng, 2018]. CornerNet detects two points of interest (the top-left and bottom right corners) of the target bounding boxes on dedicated heatmaps and then aggregate the corners to generate the final predictions (see Figure 3.4). Building on this idea, ExtremeNet [Zhou et al., 2019] and CenterNet [Duan et al., 2019] propose to add more points of interest to

[1]This score is to be mitigated as it was obtained using TensorRT optimization. Some people have reached similar results for other architectures (see this link for more details).

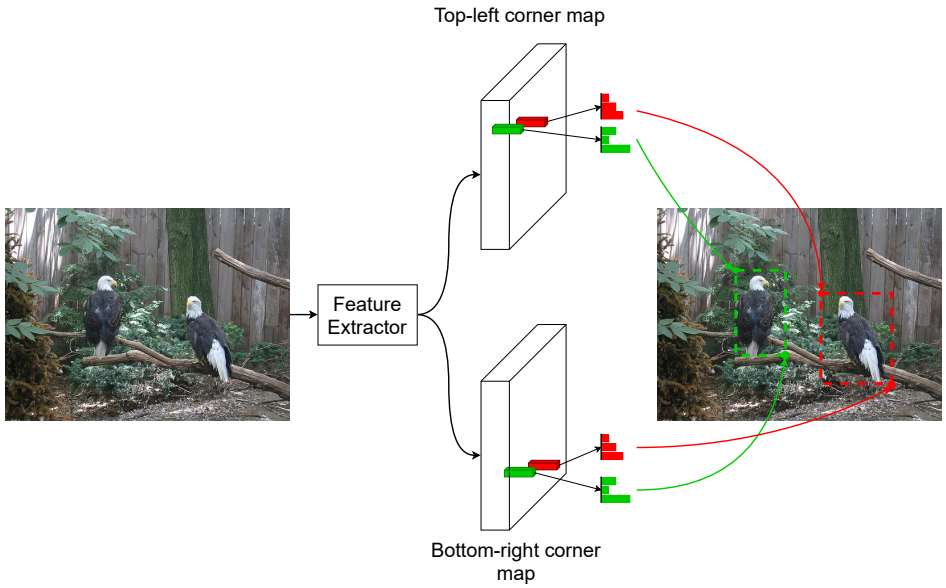


Figure 3.4 – Base architecture for anchor free detection illustrated by CornerNet [Law and Deng, 2018]. The network detects the two corners of the objects and then associates them based on an embedding. The example was taken from [Law and Deng, 2018].

robustify the aggregation step. Finally, RepPoints [Yang et al., 2019] proposes to use deformable convolution to estimate the shape of the objects to be predicted. RepPoints was then improved into RepPointsv2 [Chen et al., 2020], adding constraints on the predicted points during training.

Lastly, multiple transformers-based networks have been proposed [Carion et al., 2020, Zhu et al., 2020, Liu et al., 2021]. Although transformers were originally designed for sequence processing [Vaswani et al., 2017] they are now making the current SotA on detection challenges (Swin [Liu et al., 2021], MS-COCO). These networks completely break the classical object detection framework. Images are divided in patches and processed sequentially to output a fixed number of boxes. As this number of boxes is usually very low, transformer-based networks do not require the Non-Maximum Suppression (NMS) postprocessing, simplifying the overall processing pipeline (see Figure 3.5 for DETR [Carion et al., 2020] processing pipeline). Furthermore, on top of being effective they also manage to reach high speed performances (28 FPS). However, this score is obtained on an unknown GPU (likely V100), mitigating the interest of the results for real-time (25 FPS) predictions at large scale. Similarly to the two-shot CPN [Duan et al., 2020], the required high-end GPUs would make deployment order of magnitude too costly.

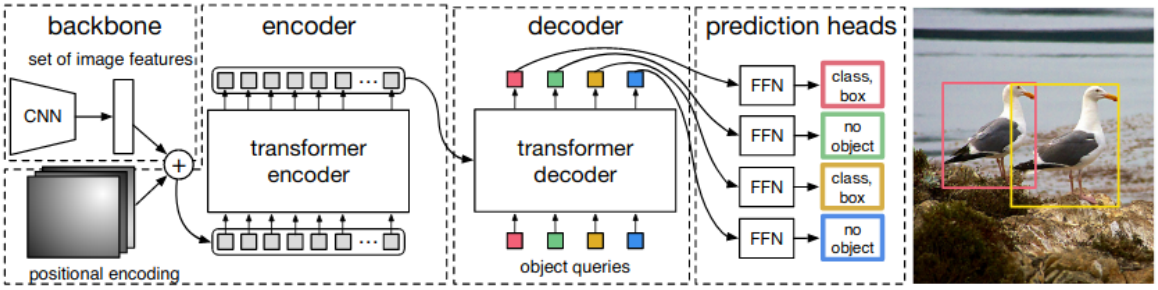


Figure 3.5 – Example of transformer based object detection (image from DETR [Carion et al., 2020]). Objects are predicted as a sequence, rather than densely from a feature map. Such pipeline reduces the number of boxes predicted and, therefore, removes the need for NMS postprocessing.

One-shot paradigm started aiming to provide with fast detection networks. They managed to do so at the cost of a reduced accuracy when compared with two-shot detectors. However, the difference in accuracy has been erased over time and one-shot detectors have overthrown two-shot detectors for both accuracy and detection speed, making them the ideal candidate for accurate real-time predictions.

Architecture	P-VOC 07	P-VOC 12	MS-COCO	FPS	test GPU
Two-shot detectors:					
R-CNN, Girshick et al. [2014]	58.5	-	-	0.08	-
SPP, He et al. [2014]	59.2	-	-	2.6	Titan
Fast R-CNN, Girshick [2015]	66.9	68.4	-	3.13	K40
Faster R-CNN, Ren et al. [2015]	73.2	70.4	-	5	K40
MNC, Dai et al. [2016a]	-	75.9	-	2.8	K40
R-FCN, Dai et al. [2016b]	80.5	77.6	-	5.9	K40
FPN, Lin et al. [2017a]	-	-	36.2	5.8	M40
Mask-RCNN, He et al. [2017]	-	-	38.2 (39.8)	5	M40
CoupleNet, Zhu et al. [2017b]	**82.7**	**80.4**	34.4	8.2	Titan X
SNIPER, Singh et al. [2018]	-	-	46.1 (47.6)	5	V100
Cascade R-CNN, Cai and Vasconcelos [2018]	-	-	42.8	7.1	Titan Xp
InterTwiner, Li et al. [2019a]	-	-	42.5 (44.2)	3.1	Titan X
TridentNet, Li et al. [2019b]	-	-	48.4	-	-
CBNet, Liu et al. [2020a]	-	-	40.8 (53.3)	6.9	-
CPN, Duan et al. [2020]	-	-	39.7/41.6 (49.2)	43.3/26.2	V100
One-shot detectors:					
YOLO, Redmon et al. [2016]	52.7/66.4	57.9	-	**155**/21	Titan X
SSD, Liu et al. [2016]	74.3/76.8	72.4/74.9	23.2/26.8	59/22	Titan X
YOLOv2, Redmon and Farhadi [2017]	69.0/78.6	-/73.4	-/21.6	91/40	Titan X
RetinaNet, Lin et al. [2017b]	-	-	32.5/37.8 (40.8)	13.7/5.1	M40
DSSD, Fu et al. [2017]	81.5	80.0	33.2	6.6	Titan X
MobileNet, Howard et al. [2017]	-	-	19.3	-	-
CornerNet, Law and Deng [2018]	-	-	40.6	4.1	Titan X
YOLOv3, Redmon and Farhadi [2018]	-	-	28.2/33.0	45.5/19.6	Titan X
RefineDet, Zhang et al. [2018]	80.0/81.8	78.1/80.1	29.4/33.0 (41.8)	40.3/24.1	Titan X
MobileNetv2, Sandler et al. [2018]	-	-	22.1	5	CPU
ExtremeNet, Zhou et al. [2019]	-	-	40.2 (43.7)	3.1	-
CenterNet, Duan et al. [2019]	-	-	44.9 (47.0)	2.9	P100
RepPoints, Yang et al. [2019]	-	-	46.5	-	-
NAS-FPN, Ghiasi et al. [2019]	-	-	37.0/48.3	26.7/3.6	P100
MobileNetv3, Howard et al. [2019]	-	-	16.1/22.0	23.3/8.4	Google Pixel Phone
MatrixNet, Rashwan et al. [2019]	-	-	42.7/44.7 (47.8)	4.0/2.8	-
EfficientDet, Tan et al. [2019]	-	-	34.6/55.1	83/3.5	Titan V
SpineNet, Du et al. [2020]	-	-	39.9/45.3 (52.1)	85.5/29.2	V100
RepPointsv2, Chen et al. [2020]	-	-	44.4/49.4 (52.1)	10.1/3.8	Titan XP
YOLOv4, Bochkovskiy et al. [2020]	-	-	41.2/43.5	96/62	V100
Scaled-YOLOv4, Wang et al. [2021]	-	-	55.5/22.0	16/**1774***	V100/2080 Ti
DetectoRS, Qiao et al. [2020]	-	-	51.3 (55.7)	3.9	Titan RTX
DETR, Carion et al. [2020]	-	-	42.0/44.9	28-10	-
Deformable DETR, Zhu et al. [2020]	-	-	46.9 (52.3)	19	V100
Swin, Liu et al. [2021]	-	-	47.2/51.9 (**58.7**)	22.3/11.6	V100

Table 3.1 – Accuracy, speed and test GPU for the existing detection methods. P-VOC denotes Pascal VOC. Accuracy results in parenthesis are the best reported results in the literature, usually not running in real-time speed due to the usage of various tricks such as multi-scale testing. The * denotes a result that was obtained using TensorRT optimization, making comparison with other networks difficult.

3.1.2 Computer vision on compressed signals

Using compressed images has been explored in the past for various computer vision tasks (such as, for instance, classification [Gueguen et al., 2018] or object counting [Wang et al., 2017]). Many applications estimating flows from videos take advantage of the compression format as the encoded data often include displacement information in order to reduce the size of the videos by exploiting temporal redundancy. Wang et al. [2018] proposed an architecture for object detection within compressed videos. They use motion vectors and residuals (c.f section 1.2 of chapter 1) to infer objects through time, while only partially decoding the compressed video flux. Wu et al. [2017] also proposed to exploit encoded motion vectors

and residuals to improve both accuracy and inference speed for a task of action recognition in videos. Taking another approach, Shou et al. [2019] detects action by first generating an optical flow from motion vectors and residuals and then using it to classify the action. We shall mention that, while not coupled with deep learning, motion vectors and/or DCT coded frames were leveraged to count vehicles Wang et al. [2017] or to estimate the vehicles' speed and density on highways Yu et al. [2002b].

Moving away from video processing, Gueguen et al. [2018] investigated different network architectures for JPEG image classification. They reached state-of-the-art classification performance while speeding-up the prediction stage. The gain for such architectures mainly stems from the reduced data transfer between CPU and GPU due to the image compression. Building on this method, several works have proposed to reduce or constrain the range of the frequency components used as input [Santos et al., 2020, Xu et al., 2020, dos Santos and Almeida, 2020, Dziedzic et al., 2019]. While [Santos et al., 2020, Xu et al., 2020, dos Santos and Almeida, 2020] only study the limitation of the number of frequency components (especially in high frequencies), Dziedzic et al. [2019] sought to constrain the frequency spectra of convolution filter for processing RGB images. Looking for networks adapted to the JPEG compression, Ehrlich and Davis [2019] develop JPEG domain equivalent of the spatial convolution and batch normalization layers. They show equivalent accuracy to RGB images on various classification datasets (MNIST [LeCun and Cortes, 2010], CIFAR10 and CIFAR100 [Krizhevsky et al.]). Lo and Hang [2019] tackle image segmentation using JPEG compressed images. They adapted an RGB network by removing the down-sampling blocks to match the shape of the DCT inputs. Impressive results almost reaching the RGB baseline with similar level of FLOPs are achieved on the Cityscapes dataset [Cordts et al., 2016]. Finally, relying on JPEG2000 compression norm, Chamain and Ding [2019] showed results matching the RGB baseline as well as speed improvements for classification. To do so, they stack sub bands of half decoded images and feed them to a modified neural network.

Overall, the usage of the compressed representation of data has drawn little attention when compared to the classical RGB processing. However, in the recent years, it seems that this peculiar topic has seen a resurgence of interest. Interestingly enough, where image processing makes usage of the compressed frequency components, video processing methods seem to mainly focus on the usage of the motion information. Aside video processing, the existing works on compressed JPEG images, and especially Gueguen et al. [2018] and [Lo and Hang, 2019] tend to demonstrate that object detection should be feasible using compressed JPEG images.

3.1.3 Synthesis

Recently, a growing body of publications tried to take advantage of the data compression to reduce the bandwidth and resources requirements. Such regain of interest for optimization can probably be linked to the growing maturity of deep learning. As performances rise, many try to adapt the advances to industrial applications, leading to a revived interest for optimization. In particular, several have tried to leverage the JPEG compression and its frequency representation to reduce bandwidth requirements ([Santos et al., 2020, Xu et al., 2020, dos Santos and Almeida, 2020, Dziedzic et al., 2019]), based on the work from Gueguen et al. [2018]. However, classification does not rely as much as object detection on spatial information. As only the image as a whole is to be classified, position becomes irrelevant. This is not the case for object detection, where each object needs to be precisely located. Therefore, unlike classification, the loss of spatial information through DCT is likely to impede the prediction results for object detection. Our works [Deguerre et al., 2019, 2021] demonstrate how to leverage JPEG compressed information to carry object detection. It is to be noted that following in our steps, others have tackle other spatial tasks such as segmentation [Lo and Hang, 2019].

Object detection in the RGB domain was largely studied over the years. Two main archi-

tectures have emerged: one-shot and two-shot detectors. Nowadays, one-shot detectors are amongst the best architectures. In particular, transformers-based architectures have shown impressive results, improving the SotA [Liu et al., 2021] and, more generally taking spots in the high ranking architectures ([Carion et al., 2020, Zhu et al., 2020]) on the MS-COCO detection task. We aim to run object detection at large scale, therefore, if we were to select an architecture for experimenting today, we would probably select Scaled-YOLOv4, with its impressive 1774 FPS after optimization. However, at the time we started the thesis (2018), the best architecture was the SSD [Liu et al., 2016], providing with a good balance between speed and accuracy. As such, our proposed method for object detection using compressed JPEG images is based on the SSD.

3.2 Object detection on compressed JPEG images

Our goal is to design an object detection network starting from compressed JPEG images. The JPEG norm is mainly based on the block-DCT transform to improve the compression ratio of images. While efficient for compression, such change of space, from pixels space to frequency domain, brings new challenges for the localization and classification of objects. Indeed, contrary to RGB images, the spatial information relative to the objects may be impeded by the DCT transform. As we aim to improve the speed of prediction so as to detect objects in real time, we lay our proposal on one-shot detection architectures and more specifically on the SSD [Liu et al., 2016]. Because the one-shot networks were built to handle RGB images, they cannot natively process JPEG compressed images. In particular, they were designed to process RGB inputs 8 times larger in width and height than their compressed counterparts. Futhermore, they were not designed to handle image components with multiple resolution (C_b, C_r subsampling) In this section, we detail how to redesign one-shot detection networks to handle these various discrepancies.

The rest of the section is divided as follow: first we detail the SSD architecture, then we explain how one-shot architectures can be modified to exploit compressed JPEG images, and, finally, we review our SSD networks newly redesigned to operate in the JPEG compressed domain.

3.2.1 Details of the Single Shot Multibox Detector

The Single Shot Multibox Detector (SSD) [Liu et al., 2016] is a one-shot detector. In its initial formulation it is composed of a pre-trained VGG backbone [Simonyan and Zisserman, 2015] and extra feature layers that act as a detection head. The SSD was developed to predict boxes at multiple scales and relies on a pyramidal architecture to do so. As the feature maps are reduced in size throughout the network, each cell within the subsequent maps gather information from a increasingly large portion of the original image (see Figure 3.6). Taking advantage of the information embedded into these cells of lower resolution (when compared with the original image), the probability of an object being present in a given cell can be estimated. The SSD uses feature maps from the size of (38,38) to (1,1) for prediction and the size of the objects being predicted is relative to the resolution of the cells. For instance, each cell of the feature map of size (38,38) is used to predict objects of size $\frac{300}{38} \approx 8$ pixels (300 being the number of pixels in the input image) while the feature map of size (1,1) predicts objects of size ≈ 300 pixels. It is worth noticing that the number of objects the network can predict per cell varies depending on the feature map and amounts to a grand total of 8732 possible objects. The whole network is detailed in Figure 3.6.

Each object that can be predicted from a given cell has a predefined box shape. These boxes are called default boxes (also prior boxes or anchor boxes) and aim to account for the variability in the shape of the objects. They are defined *a priori* and can be set manually (case of the SSD) or estimated from the training set [Redmon and Farhadi, 2017]. A representation of such boxes at various scales is shown in Figure 3.7. In the figure, the default boxes closest

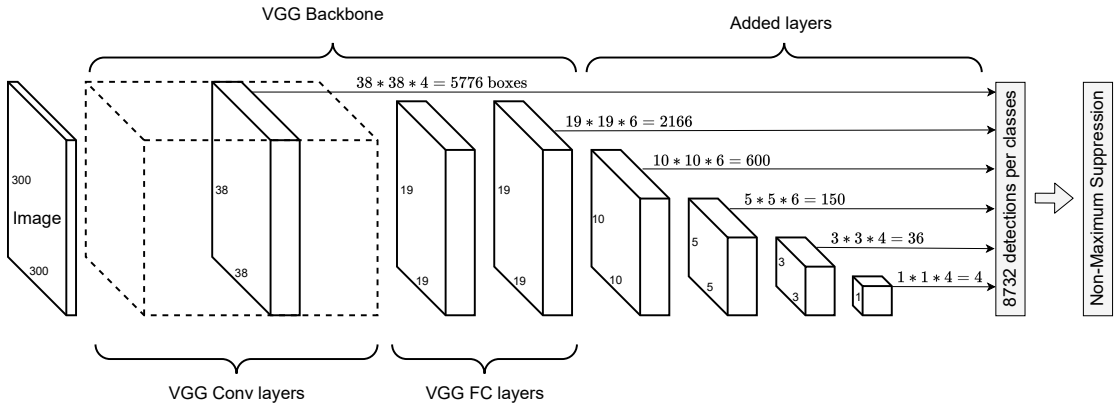


Figure 3.6 – Single Shot Multibox Detector architecture. The first part of the network is a VGG network and the second part are added layers used for multi-scale prediction. Both the convolutional layers and fully connected (FC) layers from the VGG are used for initialization of the SSD weights (FC layers are cast as convolutions). Boxes are predicted at 6 different scales for a total of 8732 boxes.

to the objects (in term of IoU) are displayed in a color identical to the objects. Although we see that the default boxes cover a large range of objects, we can notice that they do not match perfectly the ground truth bounding boxes. To avoid too approximate predictions, the SSD actually outputs the difference b between the default box d and the ground truth box g rather than only relying on the prior boxes. On top of this prediction of differences, the SSD also predicts its confidence c in each possible box to be of each possible class. As it is obvious that many default boxes will not be associated to an object, the classification layer includes a default background class. Finally, for each output feature map, a convolutional layer is applied. It produces a final map of size $(w_f, h_f, n_{boxes} \times (4 + n_{classes} + 1))$, where w_f is the width of the map, h_f its height and the part multiplied by the number of boxes corresponds to the center positions, width and height of the box plus the number of classes augmented with the background class.

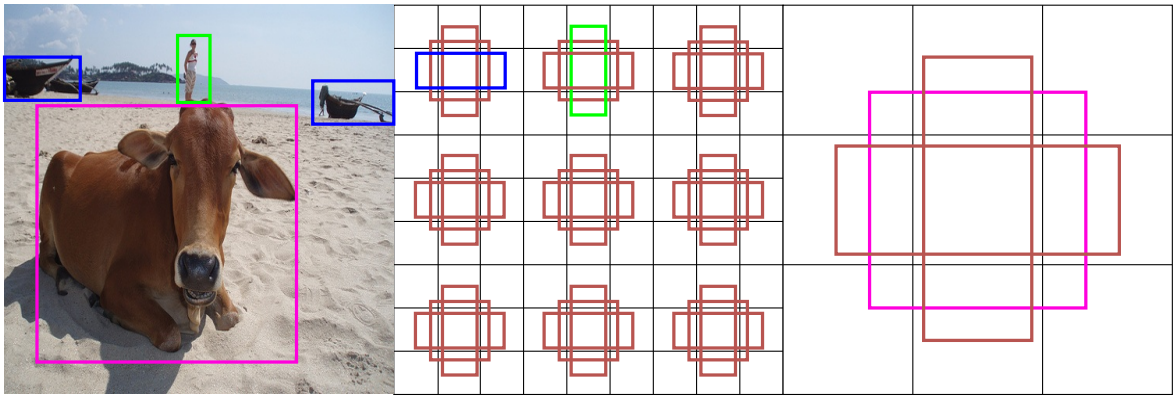


Figure 3.7 – Example of the multi-scale box prediction (feature map shapes and number of boxes do not follow the SSD architecture). On the left, the original image with the objects to be detected (a cow, a person and two boats). In the middle, a feature map of size (9,9), with three possible boxes to be predicted per cell. On the right an other feature map later in the pyramid. Detected objects are highlighted with the same color (boat on the right is not shown for clarity).

We now review the proposed loss to train a SSD network. Let define as $\hat{\mathbf{y}} = \{\hat{\mathbf{b}}, \hat{\mathbf{c}}\} = \{(\hat{b}_1, \hat{c}_1), ..., (\hat{b}_{8732}, \hat{c}_{8732})\}$ all the boxes and associated classes that are predicted by a SSD network. The SSD loss is based on the two parts (regression and classification) formulation provided in Equation 2.13. Given a set of ground truth boxes $\mathbf{g}$, we denote $\delta_{ij}^{p} = \{0, 1\}$ an indicator for matching the i-th ground truth box g_i of class p to the j-th default box d_j. The indexes j that were matched positively to a ground truth box are noted as Pos and the ones

that were not matched as Neg. Then, the classification loss is defined as:

$$L_{class}(\hat{\mathbf{c}}) = \sum_{j \in Pos} \sum_{i} \sum_{p} \delta_{ij}^{p} \mathcal{L}_c(\hat{c}_j, p) + \sum_{j \in Neg} L_c(\hat{c}_j, 0), \tag{3.1}$$

where

$$L_c(\hat{c}_j, p) = -\log \frac{\exp(\hat{c}_j^p)}{\sum_{q=1}^{n_{class}+1} \exp(\hat{c}_j^q)}, \tag{3.2}$$

$\hat{c}_j^p \in [0, +\infty]$ being the confidence of the network that box j is of class p[2]. For the default boxes that were matched with a ground truth box, the aim is to maximize the probability of the correct class p, and for the others, the probability of the background class. The second part of the object detection loss is the regression loss. It aims to measure the difference between the predicted boxes and the ground truth ones. As the SSD outputs boxes at multiple scales, the values to be predicted need to be normalized. Without such normalization, errors on large boxes may overrun the errors on small boxes. Given a ground truth box $g = (g^{cx}, g^{cy}, g^w, g^h)$, where g^{cx}, g^{cy}, g^w and g^h are respectively the location on the x-axis and y-axis of the center of the box and of its width and height, and its associated default box $d = (d^{cx}, d^{cy}, d^w, d^h)$, the normalized box $b = (b^{cx}, b^{cy}, b^w, b^h)$ is defined as:

$$\begin{aligned}
b_j^{cx} &= \frac{g_j^{cx} - d_i^{cx}}{d_i^w}, \\
b_j^{cy} &= \frac{g_j^{cy} - d_i^{cy}}{d_i^h}, \\
b_j^{w} &= \log\left(\frac{g_j^{w}}{d_i^{w}}\right), \\
b_j^{h} &= \log\left(\frac{g_j^{h}}{d_i^{h}}\right).
\end{aligned} \tag{3.3}$$

Then, letting $\mathbf{y} = \{\mathbf{b}, \mathbf{c}\}$ be the target values, the regression loss is expressed as:

$$L_{reg}(\mathbf{b}, \hat{\mathbf{b}}) = \sum_{j \in Pos} \sum_{m \in \{cx,cy,w,h\}} smooth_{L1}(\hat{b}_j^m - b_j^m), \tag{3.4}$$

where

$$smooth_{L1}(x) = \begin{cases} 0.5x^2, & \text{if } |x| < 1, \\ |x| - 0.5, & \text{otherwise}, \end{cases} \tag{3.5}$$

is used to penalize more strongly large prediction errors. Finally, the overall loss can be defined as:

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N}(L_{class}(\hat{\mathbf{c}}) + \alpha L_{reg}(\mathbf{b}, \hat{\mathbf{b}})), \tag{3.6}$$

where N is the number of matched default boxes and $\alpha > 0$ a hyper-parameter balancing both losses.

The latter formulation has one main drawback: it is very sensitive to the high number of Neg boxes during training. Although this problem has later been addressed by Lin et al. [2017b] with the focal loss, in the SSD framework, class imbalance was addressed through "hard negative mining". Hard Negative Mining (HNM) consists in removing part of the background classes to avoid overwhelming the classification loss with negative examples. In the present thesis, we used HNM similarly to the original SSD.

[2] In practice, the softmax operation usually acts as activation function of the network. Therefore $\hat{c}_j^p \in [0, 1]$ and $\sum_p \hat{c}_j^p = 1$.

3.2.2 From RGB images to object detection in the frequency domain

We now seek to design a new framework for one-shot detection networks using compressed JPEG input images. In particular, we seek to leverage the specificities of the compression norm to speed up the detection networks and reduce their memory and bandwidth requirements. The JPEG compression algorithm is made of five steps which are: RGB to YC_bC_r transform, sub-sampling, block DCT, quantization and entropy/RLE coding. As shown in Figure 3.8, the SSD can only be used natively with YC_bC_r transformed images as this is the only operation that does not change the shape of the images nor the number of channels. Past this compression step, this assessment does not hold true as the various operations reshape the image through information removal (subsampling) or change of space (blockwise DCT). Therefore, depending on the considered input, the existing detection architectures must be redesigned to take into account the specificities of the input domain. We start by discussing which compression step should be used as data input and we then detail the newly proposed architectures to fit such data.

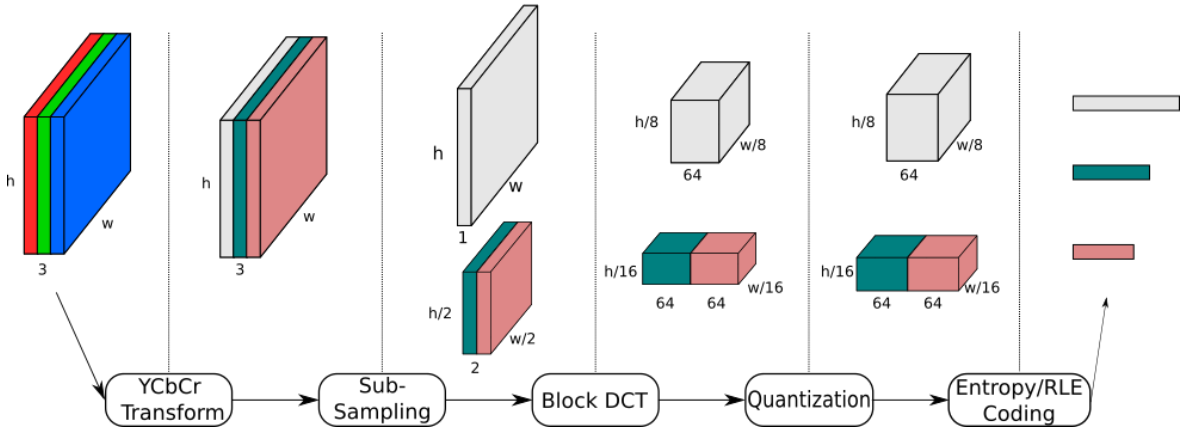


Figure 3.8 – The Full JPEG compression pipeline. The compression starts on the left with an RGB image and ends on the right with the entropy coded image. Image adapted from Gueguen et al. [2018].

Obviously we would like to avoid as much decoding steps as possible to keep the decompression requirements low. As shown in Figure 1.9, compression steps past (including) the quantization stage are problematic for data comprehension. Indeed, both the quantization step and the Huffman coding are based on tables that might change across images, potentially leading to identical representations originating from completely different uncompressed images. Therefore, we choose to use the block DCT representation as input for the detection network. The choice is bolstered by the fact that this compressed representation was used with success by Gueguen et al. [2018] for the classification task.

Let now discuss how we adapt the detection network to handle block DCT images. Compressed JPEG images have various specificities that need to be taken into account when designing a detection network. First, the size of the input image is reduced by 8 due to the blockwise DCT, second, not all the image components have the same resolution (C_bC_r are a half of Y) and finally, information is stored as frequencies rather than color pixels. We propose four different approaches that aim to account for and explore these specificities.

The simplest approach, as shown in Figure 3.9, consists in removing the first convolutional layers of the genuine SSD and in plugging the DCT input into the convolution layers with matching size. Although straightforward, the method is well fitted for object detection. Indeed, each compressed DCT block (Y component) finds itself aligned with the smallest boxes to be predicted (similarly the C_b and C_r components are aligned with the second smallest boxes). However, as shown in Figure 3.9 this approach has a major issue: the

smallest predicted boxes are not provided with the chromatic information. Therefore, we propose a second approach, where a deconvolution layer is used to scale up C_b and C_r so as to use them in combination with the Y component.

The third modification we propose can apply to the first two approaches. It adds more convolution layers at the beginning of the detection networks. This is done to provide each detection layer with a receptive field identical to the one of the RGB networks. Indeed, without this, when predicting the smallest boxes, networks in the compressed domain have a receptive field of 56 pixels while RGB networks have a receptive field of 92 pixels. It is to be noted that in order to keep the number of FPS constant when compared with the non RFA architectures, the number of filters in the first convolutional layers needs to be reduced.

Finally, because of the change in resolution induced by the JPEG compression algorithm, we assume that all the components may not be useful. Intuitively, as the sub-sampling operation is carried before the blockwise DCT, the C_b and C_r 8×8 blocks are actually a sparse representation of an equivalent four Y 8×8 blocks. Hence, the learning algorithm has to deal with both sparsity and resolution problems when the C_b and C_r inputs are merged into the network, potentially impeding the training. We thus propose a last approach that only uses the Y component as input.

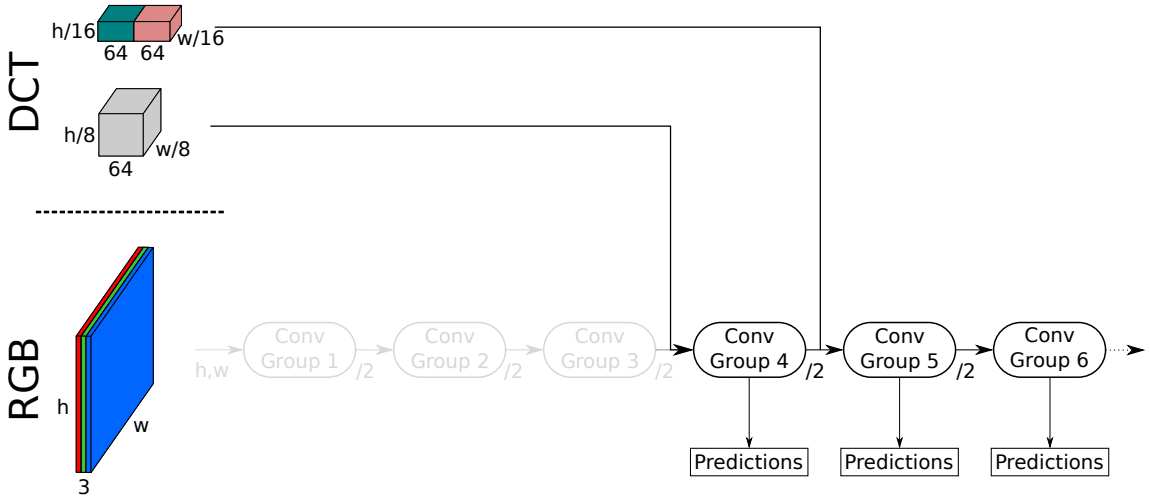


Figure 3.9 – Principle of the DCT-based object detector. The further we advance in the network, the bigger the predicted boxes. Depending on the setup, YC_bC_r or only Y inputs are fed to the networks. For clarity, not all the prediction layers are shown.

3.2.3 Proposed architectures

We now detail all of the proposed architectures. Originally, the SSD was built on top of a VGG classification network. Besides modified VGGs for frequency domain, we also investigate ResNet50-based classification backbones, as such networks have provided impressive results in terms of accuracy and inference speed for JPEG image classification [Gueguen et al., 2018]. The main characteristics of the networks are summarized in Table 3.3 and the detail of the layers of these detection architectures are summarized in Table B.1 (see Appendix Appendix B). Notice that the networks using the sole Y channel are not detailed as they are simplified instances of the YC_bC_r based networks. Similarly to Gueguen et al. [2018] we call networks with a corrected receptive field, Receptive Field Aware (RFA).

We start by reviewing the RGB architectures. Then we detail the architectures using the compressed inputs that do not use deconvolution layers. And we finish with the deconvolution-based architectures.

Conv Group	Output Size	VGG	ResNet50
conv1	112×112	$[3 \times 3, 64] \times 2$, 3×3 maxpool, s2	$[7 \times 7, 64, \text{s2}]$
conv2	56×56	$[3 \times 3, 128] \times 2$, 3×3 maxpool, s2	3×3 maxpool, s2, $\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3	28×28	$[3 \times 3, 256] \times 2$, 3×3 maxpool, s2	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 3$
conv4	14×14	$[3 \times 3, 512] \times 2$, 3×3 maxpool, s2	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 3$
conv5	7×7	$[3 \times 3, 512] \times 2$, 3×3 maxpool, s2	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
FC6	1×1	FC-4096	-
FC7	1×1	FC-4096	-
prediction	1×1	FC-1000, softmax	avg pool, FC-1000, softmax

Table 3.2 – Detail of the VGG and ResNet50 architectures. Note that the FC6 and FC7 layers can be reshaped to have a size of 7×7.

RGB baselines

For each of the backbones (VGG and ResNet50), we compare our results with the RGB-based architecture. For the VGG-based SSD, we use the original architecture [Liu et al., 2016]. In order to use the ResNet50 as backbone, few modifications must be applied to SSD. Originally, part of the SSD convolutional layers, the FC6 and FC7 layers, were designed to use the VGG dense layers' weights (see Figure 3.6 for a visual reminder). As the ResNet50 does not contain such layers, we need to modify the SSD architecture to correctly incorporate the ResNet50 backbone (a side by side comparison of VGG and ResNet50 is provided in Table 3.2). We remove all the SSD layers up to (including) the fc7 layer and replace them with the ResNet50 ones (except for the last classification layer). This way, as for the VGG, only the weights from the classification layer are not pre-loaded into the SSD.

YC_bC_r DCT methods

We present the DCT architectures that do not account for the discrepancies in resolution between the Y and C_b, C_r components. Hence, for all the architectures of this subsection, the smallest boxes only rely on the Y input for predictions (c.f Figure 3.10). The VGG-based architectures, SSD DCT and SSD DCT RFA, are modifications of the original SSD. For the ResNet50-based architectures, we select two classification networks, Late-Concat-RFA (LC-RFA) and Late-Concat-RFA-Thinner, which have shown good precision/accuracy ratios as evidenced in Gueguen et al. [2018]. The LC-RFA architectures imitate the original ResNet50 receptive field by removing the downsizing carried by some of the ResNet50 convolution layers. LC-RFA-Thinner is a lighter, hence faster, variant of LC-RFA.

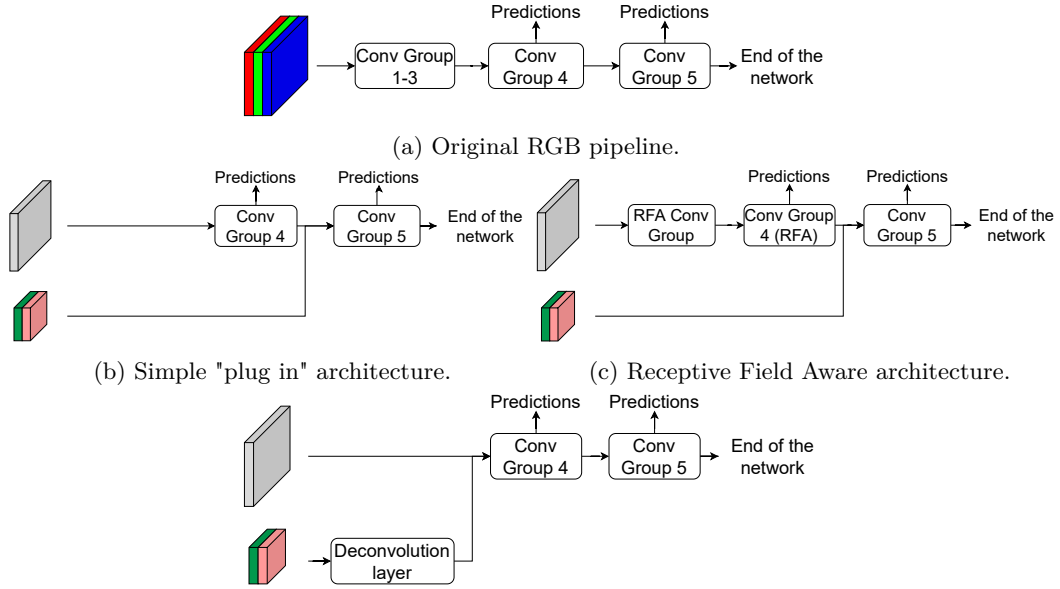
SSD DCT:

This method is based on the original SSD, the first three convolution blocks are removed and

the Y and C_b, C_r inputs are respectively plugged in the fourth and fifth block (Figure 3.10b). As the first blocks are completely bypassed, this is one of the fastest detection architecture proposed.

SSD DCT RFA:

The SSD DCT RFA is a modification of the above proposed SSD DCT network. In order to mimic the receptive field of the original RGB SSD two main changes are applied. First, three more convolution layers are added on the luminance component channel (Figure 3.10c). Second, the number of filters in the first 6 feature maps are reduced from 512 to 324 in order to keep the number of FLOPs constant with the non RFA approach.



(a) Original RGB pipeline.

(b) Simple "plug in" architecture.

(c) Receptive Field Aware architecture.

(d) Deconvolution architecture.

Figure 3.10 – Representation of the main proposed modifications. a) is the original RGB approach and b,c,d) are the proposed architectures. The Y only networks are not shown as they only require to remove the C_b and C_r components.

SSD LC-RFA:

We integrate the LC-RFA classification network in the same way as the ResNet50 SSD RGB version. This architecture does not prune away the first convolution blocks, instead, the downsizing operations are removed by changing the stride of the convolutions from 2 to 1 when required. It is worth noticing that we had to modify part the original architecture from Gueguen et al. [2018] as the one given by the authors lead to channel incompatibility on some of the layers. Specifically, we reduce the number of feature map from 1024 to 512 on the last group of convolutions for both Y and C_b and C_r axis before concatenation.

SSD LC-RFA-Thinner:

It is a lighter version of SSD LC-RFA with a reduced number of layers. This approach increases detection speed while keeping fairly good accuracy. Modifications in the number of channels is as follows: along the Y axis, channels from the three first Convolution Blocks (CB) are set from {1024, 512, 512} to {384, 384, 768} and along the C_b, C_r axis, the channels of the CB set from {512} to {256}.

YC_bC_r DCT Deconvolution methods

We use deconvolution methods to align the size of the down-sampled C_b, C_r to the one of Y (Figure 3.10d). The related architectures are given below.

SSD DCT-Deconv:
It is based on a VGG where the first three blocks are skipped. The C_b and C_r inputs first go through a deconvolution layer each and are concatenated with the Y component. Then a Batch Normalization is applied and outputs the input of the fourth block. The rest of the network follows the original SSD.

SSD Deconvolution-RFA:
It is based on ResNet50. We test using the deconvolution module proposed in Gueguen et al. [2018]. Contrary to SSD DCT-Deconv, the first blocks are not skipped, instead the stride of the first convolutions is changed from 2 to 1 when required. This architecture is mostly equivalent in speed and accuracy to the LC-RFA network for classification purpose.

Table 3.3 – Summary of the main characteristics of the proposed architectures.

Network	ResNet50-based	VGG-based	Inputs aligned	Corrected receptive field
SSD300 DCT		✓		
SSD300 DCT RFA		✓		✓
SSD300 DCT Deconvolution		✓	✓	
SSD300 DCT LC-RFA	✓			✓
SSD300 DCT LC-RFA-Thinner	✓			✓
SSD300 DCT Deconvolution-RFA	✓		✓	✓

3.3 Experiments and results

Experiments are conducted to evaluate the investigated detection networks. As a preliminary, we first implement and train the classification networks ResNet50, VGG, LC-RFA and their variants using compressed JPEG images (namely their block DCT coefficients). For this, we use the ImageNet2012 training set [Russakovsky et al., 2015]. As for detection, we train and evaluate on three datasets, Pascal VOC, MS-COCO and Actemium tunnel detection dataset.

3.3.1 Implementation details

Datasets ImageNet 2012 classification dataset [Russakovsky et al., 2015] is composed of 1,000 classes of images. The training set is made of 1,281,167 images and the validation set of 50,000 images. As the testing set is not available, we evaluate all our classification networks on the validation set, as customary amongst the community.

Pascal VOC data (object detection, [Everingham et al., 2010]) are composed of 11,530 natural scene images containing a total of 20 classes with bounding boxes for each object. We create 2 training sets by combining the data available: '07' for the Pascal VOC 2007 train-validation dataset and '07+12' for the union of the Pascal VOC 2007 train-validation and 2012 train-validation dataset. All reported results on Pascal VOC are evaluation on 2007 test set.

MS-COCO dataset (object detection, [Lin et al., 2014]) is composed of more than 100,000 natural scene images with a total of 80 classes. The training set (version 2017) is made of 118,000 images and the validation set of 5,000 images. For the evaluation, we use the provided evaluation server and the test-dev set.

Actemium dataset includes images taken from different cameras of a video surveillance system intended to monitor road traffic in tunnels in Paris (France) area. The dataset contains 3 classes (car, truck, motorcycle) with their bounding boxes and is randomly split into 1578

training images, 380 validation images and 218 test ones. The class distribution is detailed in table 3.4.

Set	Number of images	car	truck	motorcycle
training	1578	4303	658	142
validation	380	1012	143	22
test	218	588	79	29

Table 3.4 – Class distribution per set in Actemium dataset

Classification We train the classification networks using a distributed environment and follow the recommendations from Goyal et al. [2017]. The models are trained using Horovod on 4 nodes amounting to a total of 8 GPUs. Default training parameters are used for VGG and ResNet as described in Simonyan and Zisserman [2015] and He et al. [2016]. For data-augmentation, we rescale the images so that the smallest side is 256 pixels, we randomly crop a 224x224 patch and then randomly apply horizontal flip. The learning rate is decayed by 10 whenever the validation loss plateaus. While the original articles apply weight decay on the loss, due to framework limitation, we use a per layer one. We found out later that in this setting, the weight decay should actually be reduced by a factor of 2. Given that we get results close to the authors' baselines for the RGB networks, we keep the setting. The trained classification networks served as backbone for the detection networks.

We evaluate the number of Frames Per Second (FPS) that can be processed for each of the networks. We use a NVIDIA GTX 1080, set the batch size to 8 and do 10 runs of 200 predictions. The final FPS value is the average over the runs. FPS from other papers (when provided by the authors) are not reported as they used different GPUs.

Detection The SSD-based detection networks are trained on a single GPU. The networks are initialized using the weights from the corresponding classification networks.

For the VGG-based SSD, we follow Liu et al. [2016] and convert the dense classification layers into convolutional layers. When converting the VGG's dense layers weights to fit the convolution layers from the SSD, we use a pre-set sub-sampling of 0:4:4096 to extract 1024 channels from the original 4096.

For the PASCAL VOC, we train on two different sets, the original 2007 training/validation and the 2007+2012 training/validation. They are respectively denoted as 07 and 07+12 in the result tables. For MS-COCO and Actemium datasets, we use the provided training/validation sets.

We evaluate the FPS of each detection network using the same parameters as for classification. When evaluating the speed of the networks we find the Non-Maximum Suppression to be the limiting factor. For some of the architectures, this led to a sub-optimal usage of the GPU's capacities, especially for the DCT-based networks. To account for this, we report two speed evaluations: i) with one instance of the model running on the GPU, and ii) with two instances of the model running in parallel on the GPU. We stopped at two instances as the models were saturating the capacities of the GPU.

3.3.2 Evaluation of the classification networks

We rescale the smallest side of the images to 256 and keep the aspect ratio constant. We feed them to the networks and average the predictions through a Global Average Pooling layer whenever required. For each of the networks, we also retrain on RGB images to set a baseline given our data-augmentation. Results on the ImageNet validation dataset are reported in Table 3.5 and the Accuracy vs FPS is shown in Figure 3.11. We now detail the results starting with the full YC_bC_r networks and following with the Y only networks.

Network	top-1 accuracy	top-5 accuracy	FPS
State of the Art:			
VGG Simonyan and Zisserman [2015]	73.0	91.2	N/A
Resnet50 Gueguen et al. [2018]	75.78	92.65	N/A
LC-RFA (DCT) Gueguen et al. [2018]	75.92	92.81	N/A
LC-RFA-thinner (DCT) Gueguen et al. [2018]	75.39	92.57	N/A
Deconvolution-RFA (DCT) Gueguen et al. [2018]	76.06	92.02	N/A
Our trainings (VGG-based):			
VGG	71.9	90.8	267
VGG-DCT	65.5	86.4	553
VGG-DCT Y	62.6	84.6	583
VGG-DCT RFA	66.5	87.0	540
VGG-DCT RFA Y	65.0	86.2	574
VGG-DCT Deconvolution	65.9	86.7	**609**
Our trainings (ResNet50-based):			
Resnet50	74.73	92.33	324
LC-RFA (DCT)	**74.82**	**92.58**	318
LC-RFA Y (DCT)	73.25	91.40	329
LC-RFA-Thinner (DCT)	74.62	92.33	389
LC-RFA-Thinner Y (DCT)	72.48	91.04	395
Deconvolution-RFA (DCT)	74.55	92.39	313

Table 3.5 – Classification results on ImageNet. The top panel refers to the results from the literature. The two last panels refer to our implementations. In bold are the best results of our trainings.

Training with YC_bC_r DCT inputs We first retrain all the architectures presented in Gueguen et al. [2018], namely LC-RFA, LC-RFA-Thinner and Deconvolution-RFA and get accuracy results that are about 1∼2 % lower that the original ones (see Table 3.5). We attribute these differences to the fact that we evaluate on rescaled images rather than crops, as well as some possible differences in hyper-parameters as they were not fully disclosed. The main difference is related to the FPS of the networks: we do not reproduce the same speed improvements from RGB to DCT architectures. The main gains are obtained for the LC-RFA-Thinner architecture when compared with RGB with a ×1.2 (×1.77 in Gueguen et al. [2018]) speed improvement, at almost equivalent accuracy. We believe these differences are due to the different GPUs used for the evaluation. Our testing GPU does not process the images at a rate sufficient to take advantage of the reduce data transfer between CPU and GPU entailed by the compressed inputs.

We then train the VGG-DCT, VGG-DCT RFA and VGG-DCT Deconvolution networks. All the architectures perform worse than the original RGB VGG. VGG-DCT and VGG-DCT Deconvolution networks show similar performances, with the Deconvolution network slightly faster (resp. ×2.1 and ×2.2). However, while the RFA version of the networks is the slowest at 540 FPS, it is the more accurate with 67.0 top-1 accuracy.

If we compare the VGG-based networks with the ResNet50 ones, we can see that the VGG networks are about a 39% to 50% faster but reduce the accuracy by 7 to 11 points (10% to 16%). We attribute these differences to the hard pruning done on the first layers of the RGB architecture. This can be see in Figure 3.11 were all the blue dots (VGG-based networks) are at down-right position w.r.t the red points (ResNet50-based networks).

Training with only Y DCT input The related classification networks are respectively denoted as VGG-DCT Y, VGG-DCT RFA Y, LC-RFA Y and LC-RFA-Thinner Y. The obtained results (see Table 3.5) lead to the following remarks: the accuracy slightly decreases while the networks' speed increases. The smallest decrease is for the LC-RFA Y architecture

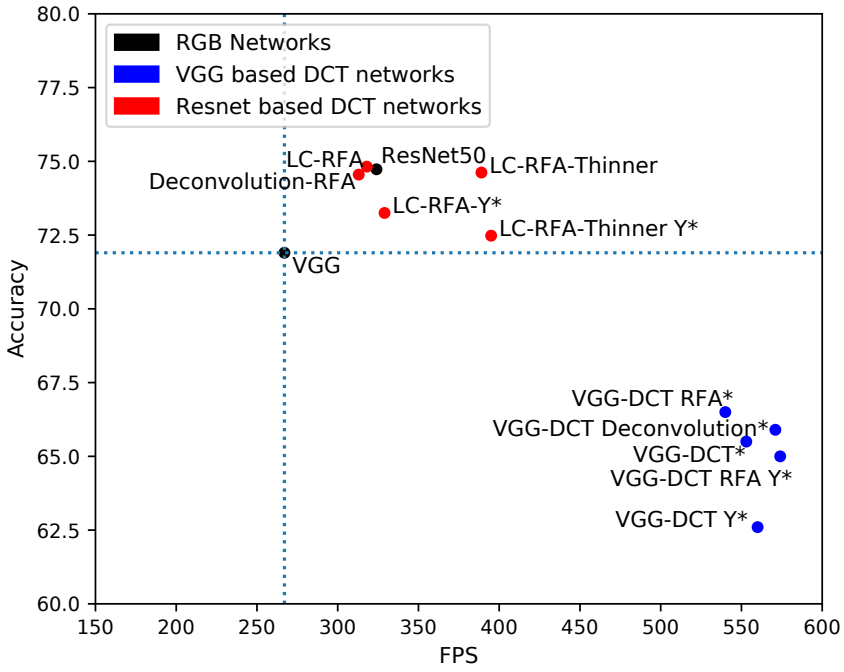


Figure 3.11 – Accuracy vs FPS for the classification networks. The starred networks are the ones modified by us.

with 3.1% drop in accuracy, while the biggest is for the VGG-DCT RFA Y network with a decrease of 8.0%. This seems to be consistent with the fact that ResNet50 classifier is more accurate than the VGG. The speed improvements, ranging from 6 to 34 FPS, are due to the reduction in computation entailed by the sole use of the Y input. While FPS gain may be negligible when comparing with the drop in accuracy, the reduced bandwidth due to the usage of the sole Y component makes such architecture attractive in case of limited computation resources.

Synthesis Overall the ResNet50 based architectures seem more robust to the use of DCT inputs, with accuracy performances similar to the RGB networks. However, this results in a reduced speed up. Where the VGG-based architectures are twice faster than the original RGB one, the ResNet50-based ones merely reach a 1.2 speed increase. This is likely due to the hard pruning carried on the VGG-based networks. As could be expected, such pruning leads to a trade off between speed and accuracy. Regarding the Y only networks, they incur a small loss in accuracy (about 1-3 points drop in accuracy) and slightly increase the detection speed (when compared with their YC_bC_r counterparts). Although such trade off might seem undesirable, the main advantage of the Y only networks lies in the fact that they require less bandwidth (about a third) for image transfer. This might suggest that images are overloaded with information as in the best case, 1/3 of the compressed image data can be dropped while only lowering the network accuracy by 3.1%.

3.3.3 Detection

We train the detection networks using the previous classification networks for weights initialization. For fairness of comparison, we also retrain the RGB networks with our trained RGB classification networks. The results for the PASCAL VOC evaluation are reported in Table 3.6, the MS-COCO results are reported in Table 3.7 and the ones for Actemium's

dataset in Table 3.9. We evaluate the speed of inference on the networks trained on the 07+12 PASCAL VOC dataset. Accuracy vs Speed is detailed in Figure 3.12 and the speed results are shown in Table 3.8. We start by detailing the results for the Pascal VOC and MS-COCO datasets and extend the discussion towards Actemium's dataset. Finally, we discuss the impact of the number of frequency components on the detection results.

Comparison of the two RGB backbones Except when training on the Pascal VOC 2007, we manage to reproduce the detection results for the SSD300 architecture (see Table 3.6). Regarding the ResNet50-based architecture, it performs worse than original SSD300 on the Pascal VOC (65.0 and 74.0 vs. 61.3 and 73.1) dataset but performs better on the MS-COCO dataset (24.5 vs. 26.8, c.f Table 3.7). These results seem to indicate that the ResNet50-based architecture has a better convergence when provided with enough training data. For both networks, FPS are mostly similar.

Network	mAP (07)	mAP (07+12)	FPS
SotA:			
SSD300 Liu et al. [2016]	68.0	74.3	N/A
Our trainings (VGG-based):			
SSD300	**65.0**	**74.0**	102
SSD300 DCT	48.9	60.0	262
SSD300 DCT Y	50.7	59.8	278
SSD300 DCT RFA	52.4	61.7	**283**
SSD300 DCT RFA Y	54.5	63.0	281
SSD300 DCT Deconvolution	38.4	53.5	282
Our trainings (ResNet50-based):			
SSD300-Resnet50	61.3	73.1	108
SSD300 DCT LC-RFA	61.7	70.7	110
SSD300 DCT LC-RFA Y	62.1	71.0	109
SSD300 DCT LC-RFA-Thinner	58.5	67.5	176
SSD300 DCT LC-RFA-Thinner Y	60.6	70.2	174
SSD300 DCT Deconvolution-RFA	54.7	68.8	104

Table 3.6 – Detection results on the PASCAL VOC 2007 test set, 07 is for trained on 2007 data and 07+12 means trained on 2007+2012 data. The last two panels report the performances of our trained networks. In bold are the best results of our trainings.

Training with YC_bC_r DCT inputs We now detail all the results for the architectures using the full compressed inputs, namely SSD300 DCT, SSD300 DCT RFA, SSD300 DCT LC-RFA and SSD300 DCT LC-RFA-Thinner. For SSD300 DCT, on the Pascal VOC 07+12 dataset (Table 3.6) we reach 60.0 mAP, 14.0 points behind the RGB method (18.9% decrease in accuracy). The RFA version of the network performs a bit better at 61.7 mAP, 12.7 points behind the RGB method (16.6% decrease in accuracy). On the MS-COCO dataset (Table 3.7), both methods lose respectively 10.2 and 9.5 points when compared with the RGB method that represent a 41.6%/38.8% decrease. As could be expected, the hard pruning of the network is a limitation factor on more complex dataset such as MS-COCO.

Regarding the ResNet50-based methods, SSD300 DCT LC-RFA and SSD300 DCT LC-RFA-Thinner outperform the VGG-based network by a large margin on the two datasets (see Figure 3.12 for a visual representation on the Pascal VOC dataset). They even outperform the original SSD on MS-COCO dataset (24.5 vs. respectively 25.8 and 25.4). While this might be expected as the used classification backbones provide similar accuracy performances, the gap is wider for detection. When comparing the SSD300 DCT LC-RFA with the SSD300 DCT LC-RFA-Thinner, we see that the second approach falls 3 points behind the first one

on the Pascal VOC (Table 3.6) datasets and has about the same accuracy on MS-COCO (Table 3.7). The main advantage of the SSD300 DCT LC-RFA-Thinner is the number of FPS it can process, ×1.63 more images than the SSD300 DCT LC-RFA while maintaining an equivalent accuracy.

	Network	Avg. Precision, IoU:			Avg. Precision, Area:			Avg. Recall, #Dets:			Avg. Recall, Area:		
		0.5:0.95	0.5	0.75	S	M	L	1	10	100	S	M	L
	SSD300 Liu et al. [2016]	23.2	41.2	23.4	5.3	23.2	39.6	22.5	33.2	35.3	9.6	37.6	56.5
VGG	SSD300 (our training)	24.5	42.4	25.2	**7.8**	25.3	38.0	23.0	34.0	35.7	**12.3**	38.1	54.4
VGG	SSD300 DCT	14.3	27.0	13.7	2.1	12.1	26.2	15.8	22.4	23.4	3.4	21.1	42.5
VGG	SSD300 DCT Y	14.4	27.0	14.0	2.1	12.0	26.5	15.8	22.2	23.3	3.5	20.8	42.3
VGG	SSD300 DCT RFA	15.0	28.1	14.6	1.9	12.9	28.1	16.3	23.0	24.0	3.1	22.2	44.1
VGG	SSD300 DCT RFA Y	15.4	28.5	15.0	2.4	13.3	28.0	16.5	23.5	24.6	4.1	22.7	43.9
VGG	SSD300 DCT Deconvolution	13.5	26.0	12.6	2.5	11.3	23.8	15.3	21.9	23.1	4.5	21.1	39.6
ResNet50	SSD300 Resnet50	**26.8**	**43.8**	**28.3**	6.2	**28.2**	**45.2**	**24.6**	**35.6**	**37.1**	10.0	**40.4**	**60.0**
ResNet50	SSD300 DCT LC-RFA	25.8	42.4	27.1	5.1	27.0	44.4	23.9	34.2	35.6	8.0	38.8	59.0
ResNet50	SSD300 DCT LC-RFA-Y	25.2	41.6	26.5	5.2	25.7	43.7	23.6	33.7	35.0	8.1	37.4	58.2
ResNet50	SSD300 DCT LC-RFA-Thinner	25.4	41.8	26.9	4.7	26.3	44.6	23.7	33.8	35.1	7.2	38.0	59.4
ResNet50	SSD300 DCT LC-RFA-Thinner-Y	24.6	40.6	25.8	4.7	24.8	43.4	23.1	32.8	34.1	7.2	36.3	57.6
ResNet50	SSD300 DCT Deconvolution-RFA	25.9	42.5	27.2	5.4	26.7	44.4	24.0	34.5	36.0	8.5	39.0	59.4

Table 3.7 – Detection results on MS-COCO test-dev set. For the precision per area, S, M and L respectively stand for Small, Medium and Large (size of the boxes). For the average recall per number of detections, 1, 10 and 100 indicate that the average recall was computed respectively given 1, 10 or 100 detection(s) per image. In bold are the best results of our trainings.

Influence of the Deconvolution on detection performance While the deconvolution networks tend to perform better for classification, we can see more mitigated results for detection. For the networks trained on the Pascal 2007 data only, we see in Table 3.6 that they have a mAP of 38.4 for the VGG-based network and 54.7 for the ResNet50-based one. They are respectively 12.3 points and 7.4 points lower than the mAP of the best performing DCT networks at equivalent speed (same backbone type, i.e VGG or ResNet). While the gap is reduced for the network trained on the 07+12 data, they still lag behind. However for the MS-COCO dataset, we can see in Table 3.7 that the SSD300 Deconvolution-RFA is the best performing of all the DCT-based architectures. Moreover, when looking at results by size of area (Small, Medium or Large), we can see that both of the deconvolution architectures improve the accuracy for small objects. Overall, it seems that when provided with enough data, the network reaches accuracy level equivalent to the non-deconvolution networks.

The SSD300 DCT Deconvolution is the fastest of all the detectors with a speed of 282 FPS but with the worst overall accuracy. The SSD300 Deconvolution-RFA has a speed equivalent to the RGB networks while not performing better than the SSD300 Resnet50.

Evaluation of the detection networks using only the Y input On the Pascal VOC dataset (Table 3.6), we get similar mAP in comparison with the networks using the full YC_bC_r input. The reverse holds true for the MS-COCO dataset (Table 3.7), where performances using only the Y input tend to be lower than their full input counter-parts (1 point below). Yet, it appears that the C_b and C_r components are not critical to correctly detect objects within images. Such result seems to be in adequation with the JPEG compression norm, which aggressively sub-samples the chroma components as they contain less critical information for our eye. Moreover, networks for detection using only the Y component have equivalent or higher speed than the networks using the YC_bC_r inputs (c.f Figure 3.12) and require less bandwidth. The SSD300 DCT LC-RFA-Thinner Y network is ×1.70 faster (with 2 instances, ×1.15 when using only one instance of the network) than the original SSD while being only 3.8 point less accurate on the Pascal VOC dataset and more accurate on the MS-COCO dataset. The SSD300 DCT Y/DCT RFA Y are even faster with a ×2.72/×2.75 speed improvement (with 2 instances, ×1.59/×1.64 when using only one instance, c.f Table 3.8)

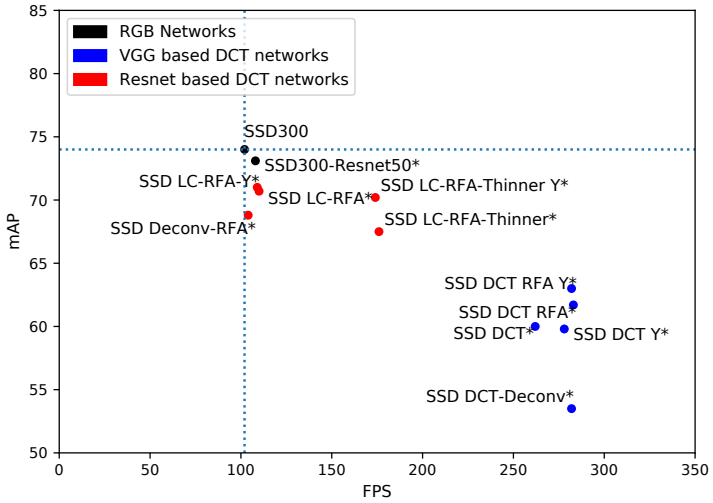


Figure 3.12 – mAP vs FPS for the detection networks on Pascal VOC dataset. Networks with a star at the end of their names are the ones modified by us.

but at the cost of 14.2/11.0 points drop in the mAP for the Pascal VOC dataset and 10.1/9.1 points for MS-COCO dataset.

On the speed of the networks The FPS ratio the networks can process are computed either with one instance or with two instances of the model on one GPU (results are reported in Table 3.8). While this may seem anecdotal, we see that most the DCT networks scale effortlessly when using 2 instances. This indicates that these architectures do not use all the computation resources available when only one network is instantiated on the GPU. However, it also indicates they can be deployed on less powerful GPUs while not incurring a loss in FPS. This is not true for the RGB-based networks, which almost use all the computation capabilities of the GPU with only one model instantiated and thus would face important loss in FPS if deployed on GPUs half as powerful. This means that the presented architectures using the compressed inputs are good matches for usage in limited resources environment or on small remote computation devices. By combining our approach with other methods, such as MobileNets [Howard et al., 2017, Sandler et al., 2018, Howard et al., 2019] or TinySSD [Wong et al., 2018], we expect to even better fit such conditions.

Extension towards vehicle detection in tunnels Table 3.9 gathers observed performances on Actemium dataset. Compared to Pascal VOC and MS-COCO, for the VGG-based architectures, the drop in mAP metrics is slight for the proposed detection model (about 5 points). This could be explained by the small number of object classes to be identified, hence allowing the SDD network to learn relevant feature maps. However, we can notice that the loss is not equally divided among the classes. The motorcycle class seems to suffer the most, with a drop of up to 16.7 points, while the cars lose in average about 3 points. Such decrease in accuracy can probably be attributed to class imbalance as the motorcycle class is largely under-represented. Still, this difference in precision between the classes could hinder the deployment of the VGG-based architecture as motorcycles are more likely to be missed while also more likely to stop on hard shoulders (to wait for the rain to stop for instance).

Regarding the ResNet50-based architectures, the results are even closer to the RGB baseline with a limited drop in accuracy. Surprisingly, the LC-RFA architectures even score 1 point higher that the RGB architecture, with the highest accuracy for the LC-RFA Y version at 86.8 mAP. However, they provide with a similar level of FPS when compared with the RGB

	Network	FPS (1 inst.)	FPS (2 inst.)
VGG	SSD300	88	102
	SSD300 DCT	136	262
	SSD300 DCT Y	140	278
	SSD300 DCT RFA	143	**283**
	SSD300 DCT RFA Y	**144**	281
	SSD300 DCT Deconvolution	**144**	282
ResNet50	SSD300-Resnet50	88	108
	SSD300 DCT LC-RFA	87	110
	SSD300 DCT LC-RFA Y	91	109
	SSD300 DCT LC-RFA-Thinner	98	176
	SSD300 DCT LC-RFA-Thinner Y	101	174
	SSD300 DCT Deconvolution-RFA	87	104

Table 3.8 – Speed inference of the tested detection networks. The tests were performed on a GTX 1080, "1 inst." means that only one instance of the model was loaded on the GPU for testing, "2 inst." means that two instances of the model were loaded on the GPU.

network (108 vs. 110 and 109). Finally, it is worth noting that the LC-RFA-Thinner Y architecture, has a loss of mAP lower than 1 point, while improving detection speed by a factor 1.7. Such impressive results make LC-RFA-Thinner Y an extremely interesting candidate to deploy in place of the RGB architectures.

Network	mAP	mAP (car)	mAP (truck)	mAP (motorcycle)	FPS
SSD300	85.6	93.0	80.9	83.0	102
SSD300 DCT	81.6	90.0	77.8	76.9	262
SSD300 DCT Y	80.2	90.4	78.5	71.7	278
SSD300 DCT RFA	78.6	90.6	78.9	66.3	283
SSD300 DCT RFA Y	84.2	92.6	81.7	78.2	281
SSD300 DCT Deconvolution	80.5	89.0	76.9	75.6	282
SSD300 Resnet50	85.4	**93.8**	78.8	80.9	108
SSD300 LC-RFA	86.3	93.0	83.1	83.0	110
SSD300 LC-RFA Y	**86.8**	91.9	81.7	**86.9**	109
SSD300 LC-RFA-Thinner	83.8	92.8	**84.2**	74.2	176
SSD300 LC-RFA-Thinner Y	84.9	93.0	81.1	80.5	174
SSD300 Deconvolution-RFA	84.0	92.4	82.2	77.4	104

Table 3.9 – Detection results for the training of the SSD on Actemium dataset.

Impact of the different DCT coefficients Finally, we experiment on the importance of the various DCT coefficients within the compressed input data. In particular, we are interested in the high frequency components, as they are supposed to contain minimal information due to the JPEG compression algorithm. To test their importance, we re-run the evaluation on the three detection datasets and only keep the X coefficients of lowest frequency (first X coefficients in the zig-zag pattern, c.f Figure 1.7) and set the others to 0. We do so with $X \in \{64, 32, 16, 8\}$ and report the results in Table 3.10.

We see that, when half of the coefficients are set to 0, the mAP only drops by 1 to 3 points on all the datasets (except for the SSD300 DCT LC-RFA-Thinner which loses 4.4 points). Surprisingly, for some of the architectures, on the Actemium dataset, the accuracy even rises when we remove the 32 highest coefficients. The SSD300 DCT, SSD300 DCT Y and SSD300 DCT LC-RFA-Thinner Y go from respectively 81.6, 80.2 and 84.9 to 81.9, 81.7 and 84.9. Such results tend to empirically confirm that the information is indeed conveyed by the lowest DCT coefficients generated by the JPEG compression. However, when more

(a) 64 coefficients kept (b) 32 coefficients kept (c) 16 coefficients kept (d) 8 coefficients kept

Figure 3.13 – Detection results (SSD DCT) in Actemium dataset depending on the number of DCT coefficients kept. As can be seen, even when only 8 coefficients are kept, the network still outputs correct detections. For clarity purposes, detection are displayed on the full resolution RGB images.

coefficients are removed, the accuracy drops more significantly on each dataset, with a loss of approximately 30% to 40% when 16 coefficients are kept and a complete breakdown when only 8 coefficients remain. Still, with only 8 coefficients, the SSD300 DCT architecture reaches an impressive 44.8 mAP on Actemium dataset. An example detection with the whole range of coefficients is shown in Figure 3.13 for Actemium dataset and more examples can be found in the appendices Figure B.2. Overall, these results show that, depending on the dataset, a large part of the DCT coefficients may be dropped without incurring a decrease in accuracy.

	Pascal VOC				MS-COCO				Actemium			
	64	32	16	8	64	32	16	8	64	32	16	8
VGG-based Networks:												
SSD300 DCT	60.0	58.1	45.1	13.4	14.4	13.5	9.6	2.1	81.6	81.9	74.8	44.8
SSD300 DCT Y	59.8	57.5	42.0	9.7	14.3	13.4	8.8	1.4	80.2	81.7	69.7	34.1
SSD300 DCT RFA	61.7	60.1	47.4	13.1	15.2	14.3	10.2	2.1	78.6	77.0	69.9	38.2
SSD300 DCT RFA Y	63.0	61.0	45.3	8.5	15.3	14.3	9.4	1.2	84.2	83.7	69.6	32.1
SSD300 DCT Deconvolution	53.5	51.6	37.1	17.2	13.5	12.9	9.1	2.6	80.5	80.1	69.4	43.6
ResNet50-based Networks:												
SSD300 DCT LC-RFA	70.7	68.4	51.1	5.5	25.7	23.7	15.4	0.8	86.3	84.4	67.6	23.4
SSD300 DCT LC-RFA Y	71.0	68.4	46.4	3.4	25.0	22.8	13.7	0.7	86.8	84.1	63.0	22.7
SSD300 DCT LC-RFA-Thinner	67.5	63.1	44.5	4.8	25.2	23.4	15.5	1.5	83.8	81.9	65.8	22.3
SSD300 DCT LC-RFA-Thinner Y	70.2	67.9	48.4	4.1	24.6	22.5	13.3	0.8	84.9	85.0	65.5	22.7
SSD300 DCT Deconvolution-RFA	68.8	67.6	53.1	12.0	25.7	24.4	16.7	1.3	84.0	80.5	68.9	40.8

Table 3.10 – Detection results on the three dataset depending on the number of DCT coefficients kept. In grey are the results with all coefficients. Note that due to submission limitations on the online MS-COCO evaluation server, the evaluation was done on the validation set rather than test set.

3.4 Conclusion

Object detection in the RGB domain has been largely studied over the years and many deep neural architectures have been proposed. However, such networks are usually not fitted for real-time large scale applications, as they require consequent amounts of computation and bandwidth resources. To circumvent this problem, we have investigated object detection in JPEG compressed images. One of the main difficulties of such task is related to the change of input domain, from a spatial domain, to frequency one. As one-shot deep detectors rely on densely predicted bounding boxes, we tackle this problem by leveraging the position of DCT blocks in the input to provide the network with spatial information.

We have devised several deep architectures based on the SSD [Liu et al., 2016] detection network framework. The architectures we explore differ in the classification backbones they rely on. Experimental evaluations evidence that they are not all equal for detection performances. In particular, the VGG-based architectures do not perform well on the MS-COCO

dataset, while the ResNet50-based architecture almost match the accuracy of the RGB network (performance drop of 5.3%) with a speed gain up to $\times 1.7$ when using compressed input. However, these results do not hold true when we test the architecture on Actemium's dataset. On this dataset, ResNet50-based DCT architectures improve the accuracy when compared with the RGB counterpart. Moreover VGG-based networks only lose about 5 points in mAP going from 85 to 80 while still maintaining a prediction speed more than twice as fast as the RGB network. On top of these results, we also empirically demonstrate that using only the Y input leads to detection performances similar to those of networks using the YC_bC_r input. The benefit is the reduced bandwidth for image transfer. Finally, we study the impact of the number of frequency coefficients on detection accuracy and demonstrate that, depending on the complexity of the dataset, nearly half of them are not required to maintain the quality of the detections. These findings are promising and may prove useful for the deployment of large real-time monitoring applications.

Although we have demonstrated the possibility to use JPEG compressed image for object detection, many questions are still to be explored. In particular, the transferability from one compressed type of input to another remains a challenge. For instance, the MPEG4-part10 compression relies on DCT blocks that can have a size of 4 by 4 pixels. If and how a network trained on JPEG images can be adapted to such inputs is an open question. Futhermore, while the JPEG compression is based on the DCT, others rely on different transforms. In particular, the JPEG 2000 compression is based on wavelet transforms. It would be interesting to see to which extent such input could be used for object detection.

Chapter 4

Object Counting in MPEG4 part-2 Compressed Videos

" Be yourself ; everyone else is already taken. "

Oscar Wilde

Contents

This chapter and the following one focus on the estimation of traffic flow parameters from MPEG4 part-2 compressed video streams. In particular, we address the task of estimating the flow rate (number of passing objects over a given duration) of vehicles moving on a fixed background.

Traffic flow theory is part of the larger field of Intelligent Transport System (ITS). It was introduced by Wardrop [1952] and aims at providing with a mathematical support for the optimization of transportation systems (for instance the minimization of the average commute time in a city). It is used from conception of the road infrastructure (maximum flow of roads, conception of intersections, etc.), to the management of built systems (changes in the road lights cycles depending on the hour of the day, display of secondary itineraries, etc.). To its core, traffic flow theory relies on a few base parameters (flow rate, density, velocity, etc.) that are used to produce an overview of the road traffic [Immers and Logghe, 2002]. Currently, these traffic parameters are estimated using induction loops. Induction loops are electromagnetic sensors installed beneath the road that activate when a vehicle passes. Such tools allow for a microscopic (per vehicle) analysis of the flow of vehicles and therefore, for a fine-grained analysis of the traffic flow in real-time. However, as they are recessed in the road, they are expensive to deploy and maintain: for each installation/maintenance operation, one needs to dig-in the road to access them. Because of these costs, induction loops are deployed at strategic areas on the road network, leaving road sections without direct flow analysis.

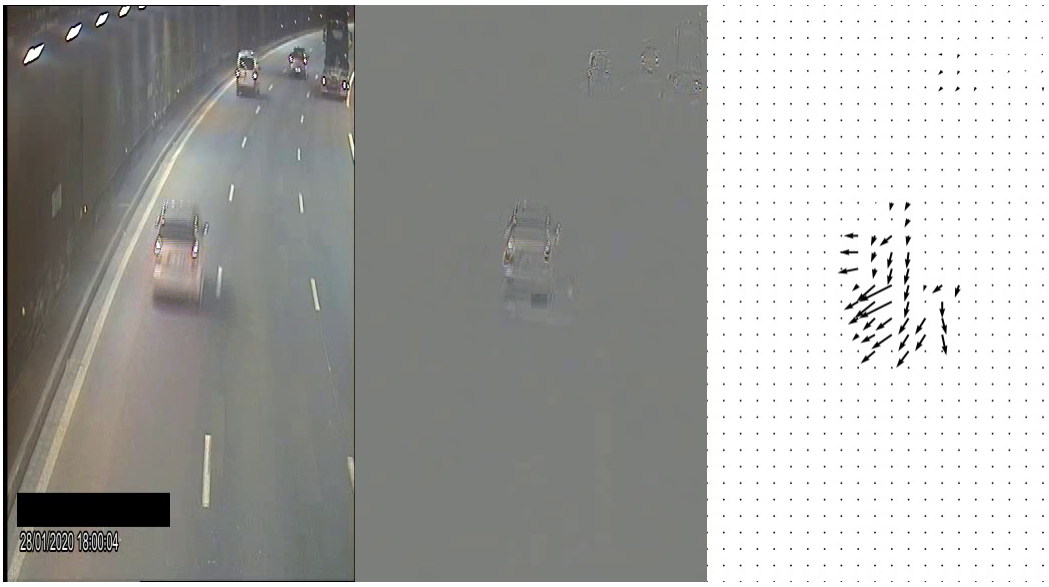


Figure 4.1 – Visualization of a frame of a video stream alongside its compressed representation. From left to right are the original RGB frame, the residual image and the Motion Vector (MV) representation. We see that only the moving vehicles generate data to be compressed. Note that motion vectors point in the opposite direction of the vehicles flow as they refer to previous frames.

While induction loops are scarcely deployed, cameras are intensively used for safety reasons: operators need to see the road in case of incidents (accidents, congestions, etc.). Therefore, the exploitation of the cameras for the estimation of traffic flow parameters would prove beneficial, allowing to cover a larger portion of roads while limiting deployment costs. With this in mind, many have tried to leverage the recorded RGB videos to provide with real-time traffic flow estimation. The first proposed approaches [Cho and Rice [2006], Schoepflin and Dailey [2007], Bernaś [2012], etc.] mainly rely on handcrafted features and complex processings. Often, such methods are not fit for large scale deployment as they usually need to be calibrated for each camera. More recently, due to the NVIDIA AI CITY challenge [Naphade et al., 2017], deep learning based methods have emerged [Kumar et al. [2018], Liu et al. [2020b], Bergmann et al. [2019], etc.]. However, as the challenge requires a microscopic (per vehicle) flow parameters estimation, all the proposed solutions rely on a tracking model

followed by a handcrafted pipeline. Furthermore, as these solutions rely on deep detectors, they are bounded in efficiency by their detectors' FPS rate (c.f Table 3.1). Hence, these methods are also not suitable for large scale deployment, as they would require a per camera calibration as well as massive computation resources, to run in real-time (in our use-case, 2000 cameras recording at 25 FPS). Moreover, the compressed data representation is completely overlooked, leaving aside potential speed up and reduction in bandwidth and computation requirements.

Traffic cameras, and more generally surveillance cameras, have the particularity that they record a fixed background, with only the objects of interest moving on screen. As video compression algorithms usually only encode differences between images, the compressed representation is interesting for flow prediction as it extracts the moving objects *per design* (c.f Figure 4.1 for an example on tunnel images). Therefore, many [Yu et al. [2002a, 2006], Li et al. [2004], etc.] have tried to leverage the compressed video data for the estimation of various flow parameters. Still, these methods heavily rely on handcrafted features, limiting *de facto* the possibilities of large scale deployment.

In such industrial context there are customary difficulties to provide with methods that generalize well to unseen data. Indeed, labelled data are often scarce due to time, resource and accessibility constraints. Moreover, even in case of available representative datasets, environment shifts may happen over time. For instance, a percentage of the cameras may be upgraded each year, leading to a change in input distribution or format. Therefore, developed solutions usually need to be adapted both to unseen data, when deployed, and, gradually, over time. Such problem may be addressed through domain adaptation [Li et al., 2021] and, as shown in chapter 5, is one of the issues faced when applying traffic flow estimation models on real data.

In this chapter, we propose a new method for the estimation of the flow rate q of objects moving on fixed background from MPEG4 part-2 compressed videos. Especially, we get rid of the pre-defined handcrafted procedures and we introduce a method that allows to learn the target task in an end-to-end fashion. We first carry a fine analysis on simulation data, such set-up allowing us to control and analyse the influence of the various cameras (according to their angle, distance to road, etc.) that can operate in real conditions (the application to Actemium's tunnel videos is developed in chapter 5). Finally, we study the possibility to adapt learnt deep models to unseen data and demonstrate the advantages and limitations of such adaptation.

The remainder of the chapter is divided as follows: first we review the existing methods and datasets available for the estimation of traffic flow parameters (section 4.1). Then we present our new method to count objects from compressed MPEG4 part-2 streams and analyze the obtained results (section 4.2). Finally, we experiment and discuss on model adaptation (section 4.3).

4.1 Estimation of flow parameters

Available methods can be divided into two main sub-groups: tracking-based (explicit) and feature-based (implicit). The tracking-based estimation follows a general pipeline that consists in detecting all the objects of interest in the stream, creating tracks from these detections, and finally, estimating the parameters from the tracks using a projection from the 2D frame space to the 3D road space. While efficient, this method has two main pitfalls. It highly depends on the detector quality, which, for deep learning-based detectors, translates into a need to manually annotate large sets of data, and, it requires to use heuristics and strong priors to compute the projections.

Contrary to the tracking-based methods, the feature-based approaches extract a set of salient features from the videos and use them to implicitly estimate the flow parameters. Unlike tracking-based methods, multiple approaches relying on handcrafted features leverage

the compressed representation of the video data (Yu et al. [2002a, 2006], etc.). Still, due to a blatant lack of data, such methods do not translate to deep learning and also heavily rely on heuristics.

Hereafter we detail the tracking-based methods and the feature-based ones[1]. Then, we present the existing datasets.

4.1.1 Tracking-based estimation

Tracking-based traffic flow estimation can be divided into two sub-groups: non deep learning and deep learning methods. While the former has, for many years, represented a vast majority of the proposed solutions, recently, the latter started to bloom mainly thanks to the yearly NVIDIA CITY challenge [Naphade et al., 2017, 2018, 2019, 2020]. Still, both methods rely on a similar pipeline of detection and tracking, the main difference being the upgrade of the detector towards a deep learning-based solution for the most recent methods.

A large part of non deep learning methods are aimed towards the surveillance of roads from Unmanned Aerial Vehicle (UAV) [Ke et al., 2017, 2019] or from high point of views [Bernaś, 2012]. Such set-up has the main advantage to avoid occlusions thanks to the position of the camera. For instance, Ke et al. [2017] carry vehicles counting through a multi-step procedure. They identify points of interest using Shi-Tomasi features [Shi and Tomasi, 1994] and use a Kanade-Lucas-Tomasi (KLT) feature tracker [Lucas and Kanade, 1981, Tomasi and Kanade, 1991] to track these points of interest between frames. Then, they cluster the points of interest based on their computed speed and direction so as to detect vehicles and use the detections for counting. Ke et al. [2019] detect objects using Haar cascade and a classification network, then a KLT tracker is used to generate the tracks in order to estimate flow density and speed. Using Haar like features, Bernaś [2012] detects and tracks vehicles from high view points by relying on a Lucas-Kanade (LK) tracker [Lucas and Kanade, 1981] and uses the tracks to count objects. Moving away from vehicle flow estimation, few works have also tackled the problem in the context of pedestrian surveillance. Lee et al. [2007] carry foreground segmentation and use optical flow information to count the pedestrians crossing a predefined line. And, using thermal images, Lahouli et al. [2018] extract Region of Interest [RoI] through various processing steps and also use the optical flow to track pedestrians.

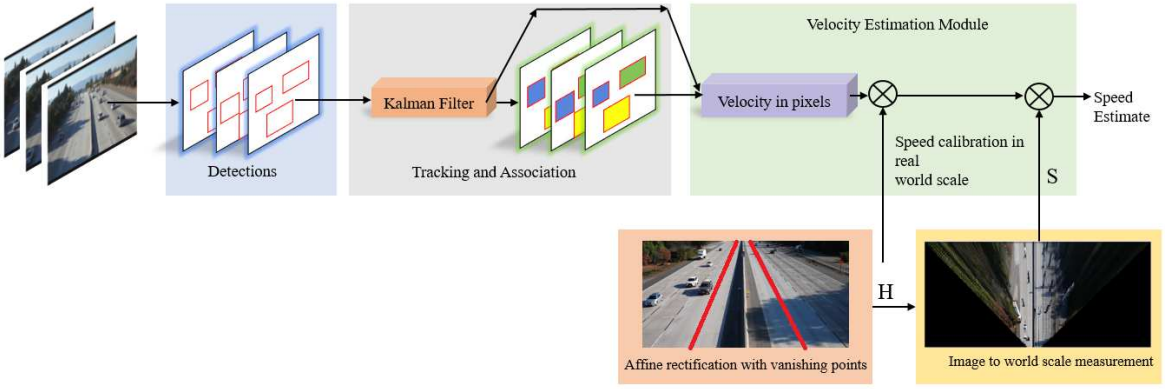


Figure 4.2 – Example of a typical processing pipeline. This peculiar example was taken from [Kumar et al., 2018]. We can see the three main steps used to process the images before estimation: detection, tracking and projection.

Recently, newer methods based on deep learning have emerged. However, due to the scarcity of data, they are mostly related to the NVIDIA AI city challenge [Tran et al., 2018, Tang et al., 2018, Shi et al., 2018, Kumar et al., 2018, Liu et al., 2020b, Bergmann et al.,

[1]Note that, while we aim for video processing, and, as such, focus on video-based articles, there are multiple ways to predict traffic flow parameters (GPS data, partial loop detectors data, etc.). However, these methods are out of the scope of the thesis.

2019, Yu et al., 2020b, Chang et al., 2020, Bui et al., 2020]. Most of these methods hinge on closely related steps. As the challenge evaluation is based on the vehicles tracking, as well as the microscopic velocity estimation, atypical approaches are implicitly discouraged. All the methods apply the same global scheme of a deep detector followed by multi-object tracking, 2D to 3D projection and estimation (such as depicted in Figure 4.2). Mainly, the various solutions change details in the processing pipeline. For instance, the detection network might change: Faster R-CNN [Tran et al., 2018, Liu et al., 2020b, Bergmann et al., 2019], Mask R-CNN [Kumar et al., 2018, Shi et al., 2018, Yu et al., 2020b, Chang et al., 2020], YOLOs [Tang et al., 2018, Bui et al., 2020]. Or, the tracking algorithm used can variate: median flow [Shi et al., 2018], hungarian matching [Chang et al., 2020], DeepSORT [Bui et al., 2020, Liu et al., 2020b]. Or, the solution used for the projection-estimation can differ: areas from landmarks [Tran et al., 2018], vanishing point estimation [Shi et al., 2018, Kumar et al., 2018], intersections crossing [Liu et al., 2020b, Yu et al., 2020b, Chang et al., 2020]. Stepping away from the AI City Challenge, Brkić et al. [2020] use a Faster R-CNN to detect vehicles from an UAV and then estimate traffic flow parameters based on manually computed lane lengths. [Li et al., 2021] tackle the problem of day to night adaptation by using a training pipeline (for the detector) based on CycleGan [Zhu et al., 2017a]. They then rely on a KLT tracker and salient road points to estimate the velocity of vehicles.

Finally, let mention that few research works have proposed to detect objects using compressed videos. For instance, Tusch et al. [2012] extract and cluster groups of MVs to compute handcrafted features that are then provided to a gaussian radial basis function network to classify the level of service (free flow, congested, etc.). Also leveraging MVs to carry object detection, Kas et al. [2009] adds RGB information to avoid mixing up multiple vehicles which can then be tracked and counted.

4.1.2 Estimation from video stream parameters

Apart from the detect and track framework, another trend of methods directly extracts features to predict the traffic flow parameters. These methods can be divided into two groups: RGB-based and compressed-representation-based.

Part of the RGB-based methods lies on the usage of intensity profiles to determine the speed of vehicles [Cho and Rice, 2006, Schoepflin and Dailey, 2007]. The main idea is to segment road lanes with the presence or absence of vehicles and to compute the shift between two segmentations at consecutive time steps. Knowing the ratio number of pixel/road distance, one can then estimate the mean speed of vehicles. Other RGB-based methods tackle density estimation and vehicle counting. For instance, Sun et al. [2019] extract various salient features such as edges or textures to predict density through regression. Zhang et al. [2017] jointly address the tasks of vehicles counting and density estimation. Their approach based on a learnt Fully Convolutional Network (FCN) is solely tested on images rather than videos. Relying on the computation of a mean background image, Zhou et al. [2021] extract changes in new frames and then estimate the traffic flow from the ratio of foreground over background pixels.

Although RGB-based methods are efficient, they do not leverage the flow information encoded in videos due to the compression algorithms. In particular, the MV frames and the residual images, that respectively encompass flow and texture information, can be used. For instance, Yu et al. [2002a] and Yu et al. [2006] extract the MVs from the video flux and filter them based on their intensity and the texture information of the DCT coefficients. Then, they project them from the 2D frame space into the 3D road space to estimate the speed and density of vehicles on the road, based on the number and intensity of the MVs. Also using both MVs and residual information, Li et al. [2004], handcraft a multi-dimensional feature vector (based on multiple variables such as MVs mean, std, etc.) which is fed to Gaussian Mixture Hidden Markov Models to estimate the traffic state (stopped, empty, etc.). Ignoring part of the compressed information, other methods solely rely on the MVs to estimate the traffic

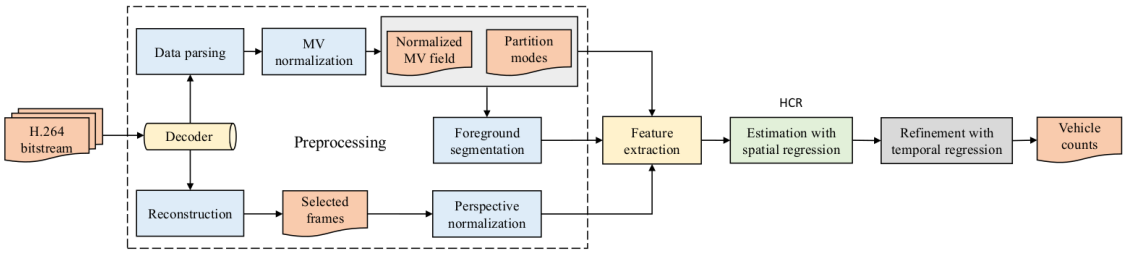


Figure 4.3 – Example of typical processing pipeline in the compressed domain. This peculiar example is taken from [Wang et al., 2019]. Images are preprocessed using multiple handcrafted modules, features are extracted and then the flow parameters are estimated.

flow parameters. For instance, Fu et al. [2009] use the MVs for calibration of the camera (detection of the area to process in the frame) and to estimate the mean velocity. Wang et al. [2019] extract various salient features (area, perimeter, shape of MVs blobs, histogram of MVs, etc.) using a complex pipeline (c.f Figure 4.3), and then estimate the number of vehicles on the road by a regression model. Finally, using both RGB and motion vectors information, Mbonye and Ferrie [2006] leverage the MVs to correctly position a controllable camera and then, estimate the traffic flow parameters in the RGB domain.

4.1.3 Datasets in the wild: traffic videos

Let us now review the existing traffic videos that can be used for our concern. We focus on datasets with a fixed background and therefore, exclude all videos that were recorded from moving objects, such as cars.

WebCamT, introduced by Zhang et al. [2017] contains about 60,000 frames, recorded at a low unknown frame rate. Each frame is annotated with the following information: bounding boxes, vehicles type, vehicles orientation, vehicle density and weather. The dataset is divided into training and testing sets, with 45,850 and 14,150 frames, respectively. Although the dataset might seem satisfactory, the recording frame rate is to low to make it possible to estimate flow parameter (c.f Figure 4.4 illustrating two consecutive frames).

(a) Frame n°1 (b) Frame n°2

Figure 4.4 – Two consecutive frames on the WebCamT dataset. Comparing the timestamps on top of the frames, we can see that they are 24 seconds apart. Therefore, the frame rate is too low to produce any usable video.

The highD dataset [Krajewski et al., 2018] is made of aerial views of german highways. The dataset has recordings of 6 different locations for a total of 16.5 hours. Recordings were done from drones flying over the highways. Although simplistic, due to the fixed top position for the recording, this dataset is one of the few very interesting for traffic flow parameters estimation. However, we were not granted access to it.

The AI CITY Challenge is a yearly challenge proposed by NVIDIA [Naphade et al., 2017, 2018, 2019, 2020]. Multiple tracks are available and are all related to traffic analysis. The 2020 tracks were: vehicle counts by class at multiple intersections, city-scale multi-camera vehicle re-identification, city-scale multi-camera vehicle tracking, traffic anomaly detection. Although seemingly fitted for the estimation of traffic flow parameters, the provided data does not contain any useful labels. The only labels available are the Regions Of Interest (RoIs) and the Movements Of Interest (MOI). Data used during the challenge was annotated by the participant and not shared. Therefore this dataset is also of limited use for the estimation of traffic flow parameters.

Suburban Traffic on GRaphs using CamEra NETworkS (STREETS) was introduced by Snyder and Do [2019]. It regroups images that are the closest to Actemium's operation conditions (surveillance cameras with limited number of vehicle flows). However, data was recorded at a frame rate of one frame per ten minutes. While 4 millions images over a period of 2.5 months across 100 distinct cameras were collected, the data cannot be used for video traffic flow estimation.

Finally, the UA-DETRAC Benchmark Wen et al. [2020] consists of sequences of images of road traffic at 24 different locations, with 8250 manually annotated vehicles. The sequences can be converted to videos, however, the total number of obtained datapoints after conversion would be of 300. Such number is too low for proper usage of deep learning. Furthermore the flows of vehicles are quite complex with urban crossings, greatly complicating the task when compared with highways or tunnels.

4.1.4 Summary

Due to its many applications, the estimation of traffic flow parameter has been largely studied over the years. Methods for estimation can be roughly divided into two main groups: detector-based and feature-based.

In their early years, detector-based methods relied on handcrafted feature to detect vehicles. The detected objects were then used to generate tracks, that were, in turn, used to estimate traffic flow parameters. More recently, mainly thanks to the yearly NVIDIA AI CITY Challenge [Naphade et al., 2017, 2018, 2019, 2020], detectors were replaced with newer deep learning methods. However, due to the design of the challenge itself, the proposed solutions still rely on a complex detection-tracking-estimation pipeline. As such methods require careful calibration, they are not suited for large scale deployments. Furthermore, the detectors used (Faster R-CNN, Mask R-CNN, YOLOv3) are not best suited for real time analysis due to their FPS level (c.f Table 3.1). Finally, as it is common for deep learning-based methods, the proposed solutions completely overlook the available compressed representation of the data and focus on the usage of the RGB representation.

Taking another approach, some avoided the detection-tracking-estimation pipeline by directly estimating the traffic flow parameters from carefully handcrafted features. Usually, such methods focus on a macroscopic estimation of the traffic flow parameters as the lack of detection step prevents from a per vehicle analysis. Part of the proposed methods are only based on RGB inputs and optical flow. However, as the video compression algorithms encode flow and texture information, others improved this formulation by leveraging the compressed data representation. While astute in their usage of available information, these methods also rely on handcrafted pipelines and are therefore not suited for large scale deployments.

Overall, the existing literature does not provide with methods that can be trained in an end-to-end fashion. Such gap in the existing solutions can be related to the lack of datasets fit for such task. Indeed most of the proposed datasets are either too small, too difficult to obtain or simply not suited for video processing. This fact is, most likely, due to the difficulties to annotate videos with flow information when induction loop data is not available.

As such, we aim to solve both problems by proposing an end-to-end method for the estimation of moving objects' flow rate on a fixed background using compressed data rep-

resentation. And, because data are not available, we propose to work on two datasets, a generated one (detailed in this chapter) and one obtained from real tunnel data through the coupling of videos and induction loops information (detailed in chapter 5).

4.2 End-to-end learning in the MPEG4 part-2 compressed video domain for flow rate estimation

We now detail the proposed method to estimate flow rate from compressed video streams. We aim to provide with a generic deep learning-based method that can correctly estimate the flow rate q from any camera and can scale up seamlessly. In particular, we focus on the usage of compressed MPEG4 part-2 video stream as Actemium's data is compressed in this format. We devise two main approaches: first we propose deep networks that estimate q through regression, then, we demonstrate how the problem of counting vehicles can be transformed into a classification problem, using a sequence alignment strategy. Moreover, as training data is scarce we evaluate the proposed methods on a toy dataset coined as: Moving Digits. This dataset, coarsely simulating traffic flow, allows to control the size of the objects, as well as the angle and the number of object flows. Its main purpose is to allow to thoroughly evaluate the proposed methods to assess their strengths and weaknesses.

The rest of the section is divided as follows: we first detail the problem, we then present our new method for traffic flow parameters estimation, after which we introduce the Moving Digits dataset and, finally, we detail experimental results.

4.2.1 Problem statement

In Paris's tunnels, about 2,000 cameras record the flows of vehicles. The long term goal is to use these video streams to produce a *real-time* analysis of the traffic flow parameters, so as to get rid of the costly counting induction loop sensors. In particular, we focus on the estimation of the flow rate q which is defined as the number of vehicles ΔN that have crossed a given road section during time interval ΔT (from 20 seconds up to 5 minutes in practice).

When compared with induction loop sensors, video analysis is a much more complex task subject to numerous degrees of freedom. Indeed, where induction loops are simple on/off sensors, video analysis suffers from camera orientation changes and illumination due to vehicle lights. Moreover, where one induction loop is used per lane, often a camera records multiple lanes and is therefore subject to occlusions. Also, there is the coarseness of the annotations. Usually, only one flow rate value is associated to a given set of frames (for instance a 20 seconds long video with an associated single flow rate value q). Such lack of annotation renders difficult the learning of a prediction model based on the microscopic identification of vehicles. Furthermore, the reduced scale of labelled video streams prevent from covering all these variability issues so as to evaluate the generalization abilities of proposed solutions.

Let us start by formally defining our flow rate estimation problem. We abstract from the vehicle counting to a more general object related formulation. Let S be a set of training examples drawn from a distribution $\mathcal{D}_{\mathcal{X}\times\mathcal{Y}}$. The input space $\mathcal{X} \in (\mathbb{R}^{H\times W\times C})*$ is the set of all possible sequences (a sequence is denoted by the $*$ symbol) of video frames (compressed or not) of height H, width W and with C channels. The output space $\mathcal{Y} \in \mathbb{R}_+$ is the set of flow rate values q (we consider the average value in case of multiple object flows) to be predicted. Each example in S consists of a pair $(\mathbf{x}, y)$. In general, the sequence $\mathbf{x}$ is of length L (in seconds), over which the value q is computed, in our case, $L = 500$. We also define $l \in [1, L]$ as the frame index of a given input sequence. As y refers to the passing objects over the L frames, we make the assumption that each sequence is statistically independent from the others. Then, the aim is to build a regression model h such that:

$$y = h(x_L, \ldots, x_l, \ldots, x_1) + \epsilon = h(\mathbf{x}) + \epsilon, \tag{4.1}$$

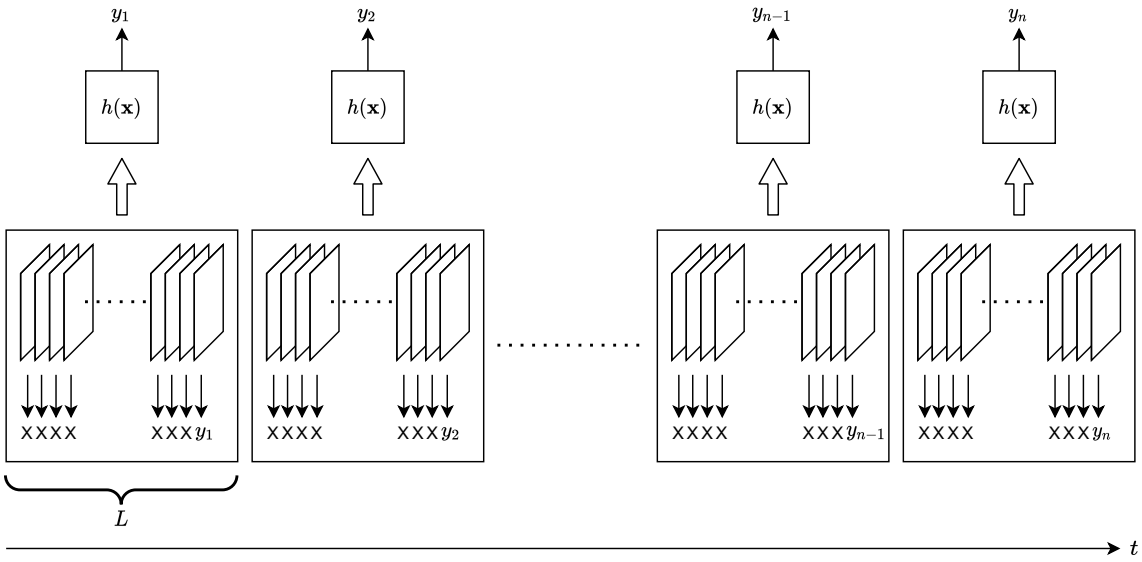


Figure 4.5 – Illustration of the problem formulation. Given sequences of L frames, we aim to predict the related flow rate y. Note that we suppose that subsequent sequences are independent. Such assumption is debatable as following sequences are likely to have similarly distributed flow rates.

where ϵ is the measurement error. The overall problem formulation is illustrated in Figure 4.5. Hereafter, we will design h as a deep network trainable in end-to-end manner. It is important to note that h must not require high computation and bandwidth resources as we aim to produce a highly scalable solution.

4.2.2 Regression Approaches

Equation 4.1 addresses the counting problem by analyzing each frame within the targeted time interval. Naively using the plain RGB videos will lead to cumbersome deep networks, not suited to large scale deployments. Indeed, processing RGB input frames requires to stack convolution layers to extract meaningful information from each frame. Hence, this may induce high memory and computation footprints.

Rather, we seek to leverage the compressed video streams in order to benefit from a more compact representation as input. The MPEG4 part-2 norm is based on a multi-step pipeline that uses Motion Vector (MV) frames and Residual frames to compress videos. MV frames contain a coarse representation of the flow of pixels between frames. They compress the information by providing the motion information for blocks of pixels (16×16 for the MPEG4 part-2 compression) rather than pixels alone. Thus, using MV frames instead of RGB frames allows to reduce the dimensionality of the input by a factor 256. Residual frames contain the difference between frames. Although they are sparse due to the removal of the background, residual frames are identical to RGB frames, both in shape and type of data (textures). As such, Residual frames, if used in replacement of RGB frames, would not solve the deployment issue. Therefore, we choose to base our proposed networks solely on sequences of MV frames.

We now devise models that can process MV frames to predict the associated q values. Such processing requires to jointly analyse spatial (frame) and temporal (sequence) information. Various layers can apply: 2D convolutions for spatial processing, ConvLSTM for temporal processing or 3D convolutions for spatio-temporal processing. Therefore, we devise three models that exploit different types of deep learning layers and are summarized in Figure 4.6:

1. *DeepMotionCLF* (short for DeepMotionConvLstmFrame): the input sequence is processed frame per frame through 2D convolutions for spatial information processing and a ConvLSTM layer to capture the temporal dependencies.

2. *DeepMotionCLS* (short for DeepMotionConvLstmSecond): in this network, inputs are grouped by seconds, each second is processed using 3D convolutions and the resulting representation is analysed using a ConvLSTM layer.

3. *DeepMotion3D*: in this architecture, the input is considered as a whole 3D signal and processed only using stacked 3D convolutions.

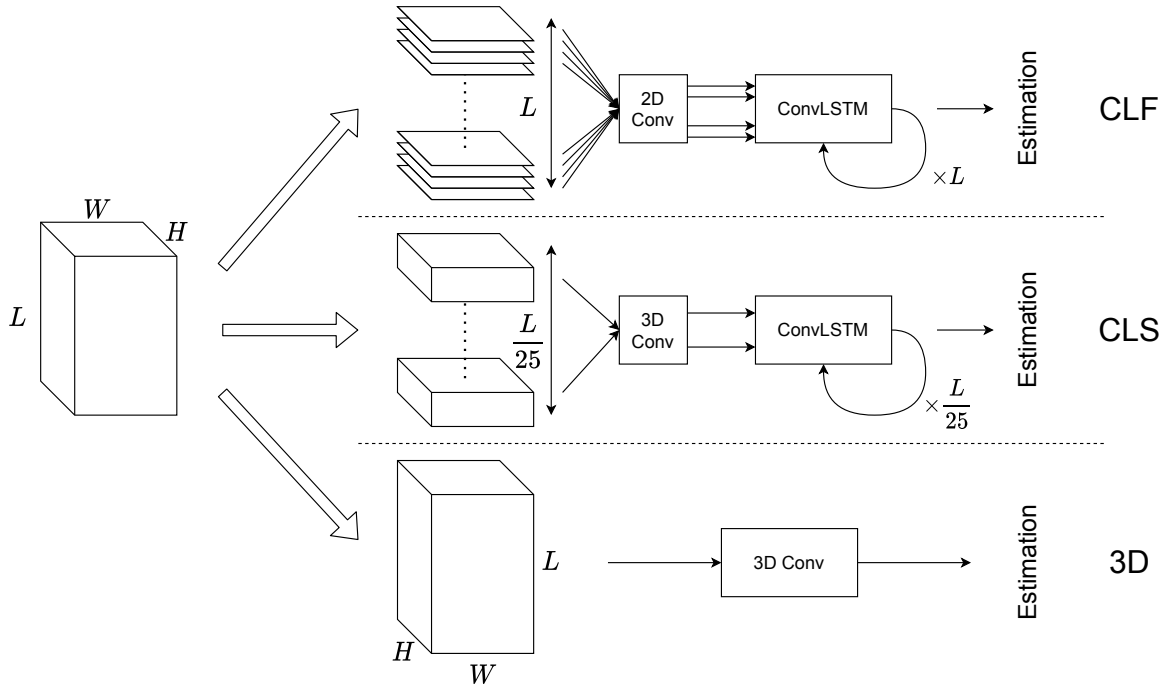


Figure 4.6 – The three proposed approaches for the estimation of the flow rate through regression. On top is the *DeepMotionCLF* approach, middle is *DeepMotionCLS* and bottom is *DeepMotion3D*.

The first approach, *DeepMotionCLF* considers the problem in its most basic formulation. At each time step, a frame representation is extracted, and then, each representation is temporally processed. 2D convolutions are the most obvious tool for the spatial information processing and RNNs for the temporal processing. More precisely, as we handle sequences of images, we base our network on ConvLSTMs [Shi et al., 2015]. The overall pipeline is shown in Figure 4.6, top panel. This approach may face shortcomings. In our case, the network has to model a long temporal dependency, as the input sequence typically covers 20 seconds. It needs to memorize the passage of each object to take it into account for the total count. Such task can prove difficult has demonstrated by Vecoven et al. [2020]. Moreover, this architecture limits the parallelization capabilities on the GPU. Finally, we can question the use of a per frame analysis of the data, given that an object takes more than a frame to cross the counting section of the images.

In order to circumvent these problems, we propose to process the input data second by second rather than frame by frame. By doing so, the ConvLSTM layer has to process fewer time steps, consequently limiting the effect of both lack of parallelization and long-term dependencies. For this approach we keep the ConvLSTM layer for temporal processing. However, as the unit component is now formed of 25 consecutive frames, we replace the 2D convolutions by 3D convolutions. This reduces the limitations on parallelization. We name this architecture *DeepMotionCLS*. The approach is detailed in Figure 4.6, middle panel.

So far, the first two approaches have considered the input as a sequence. However, we can see in *DeepMotionCLS* that the sequence size is partially reduced by grouping frames per second. Such idea, when pushed to its limit, consists in considering the input sequence $\mathbf{x}$ as a whole and processing the L frames altogether through stacked 3D convolutions. On the

one hand, such approach loses the capability to output a flow rate value at each time step (whether it is a frame or a second), but, on the other hand, it maximizes the use of the GPU by fully leveraging the parallelization capabilities. We name this approach *DeepMotion3D*. It is detailed in Figure 4.6, bottom panel.

4.2.3 Temporal classification approach

The formulation Equation 4.1 considers the problem as a regression problem. This approach is mainly dictated by the coarseness of the annotation, preventing the microscopic detection of each object. However, this lack of intermediate labels can be tackled by considering the problem as an original temporal classification problem rather than a regression. Let us consider the simple case of flow rate prediction given, one flow of objects. If $y = 8$ objects counted in the stream, we know that a series of 8 objects "oooooooo" have passed by, but we do not know when. Considering a single object class (as the flow rate label does not allow to distinguish between types of objects), the CTC loss can be used to learn both the detection and the segmentation similarly to an induction loop (on/off). In this formulation, the output y becomes a sequence $y*$ of length L, such as $y_l \in \{\bar{o}, o\}$ is a binary label. The idea is to represent the output in the same way as would an induction loop, $y_l = o$ if a object is over it, $y_l = \bar{o}$ else. It is to be noted that such approach can be used on videos with multiple flows of objects, by increasing the number of outputs to match the number of flows and using one CTC loss per flow.

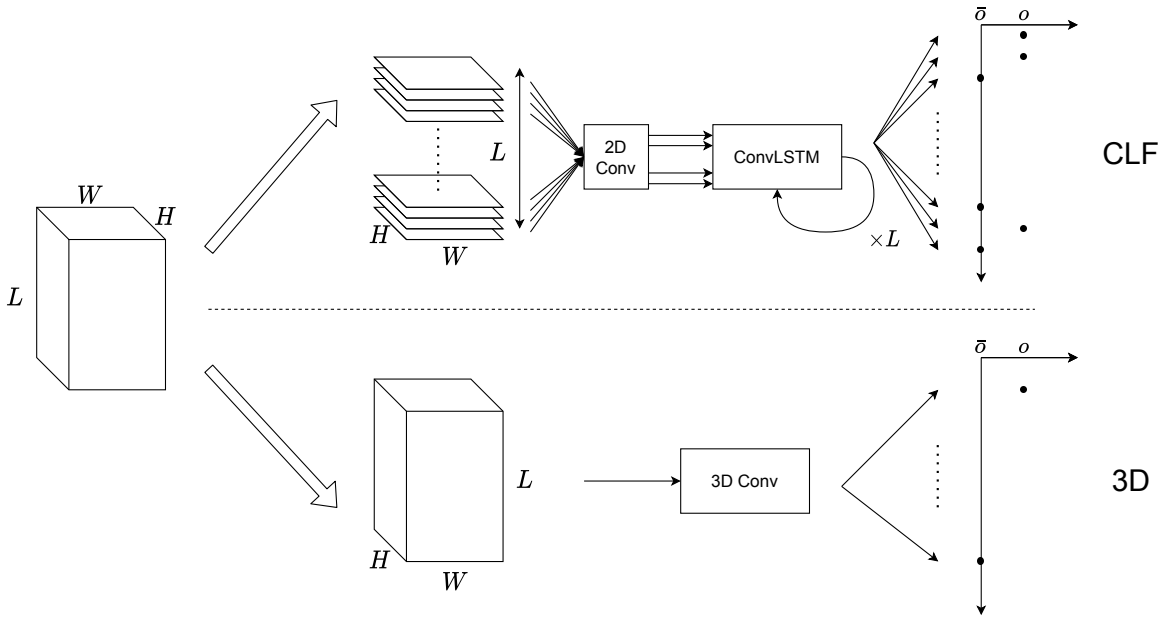


Figure 4.7 – The two proposed approaches based on the CTC loss. It is to be noted that only the prediction for one flow of objects is shown (in case of M flows, M outputs have to be considered). o and $\bar{o}$ stand for the presence/absence of objects.

Apart from the output, the processing is overall very similar to the one of the regression setting, and, as such, we devise two architectures based upon the previous ones (see Figure 4.7):

1. *DeepMotionCLF-CTC*: Similarly to *DeepMotionCLF*, each frame is processed separately by a 2D convolution and then a ConvLSTM layer is applied for temporal processing.

2. *DeepMotion3D-CTC*: in this architecture, only stacked 3D convolutions are used, then, the output tensor is split along the time axis to provide with an output for the CTC loss.

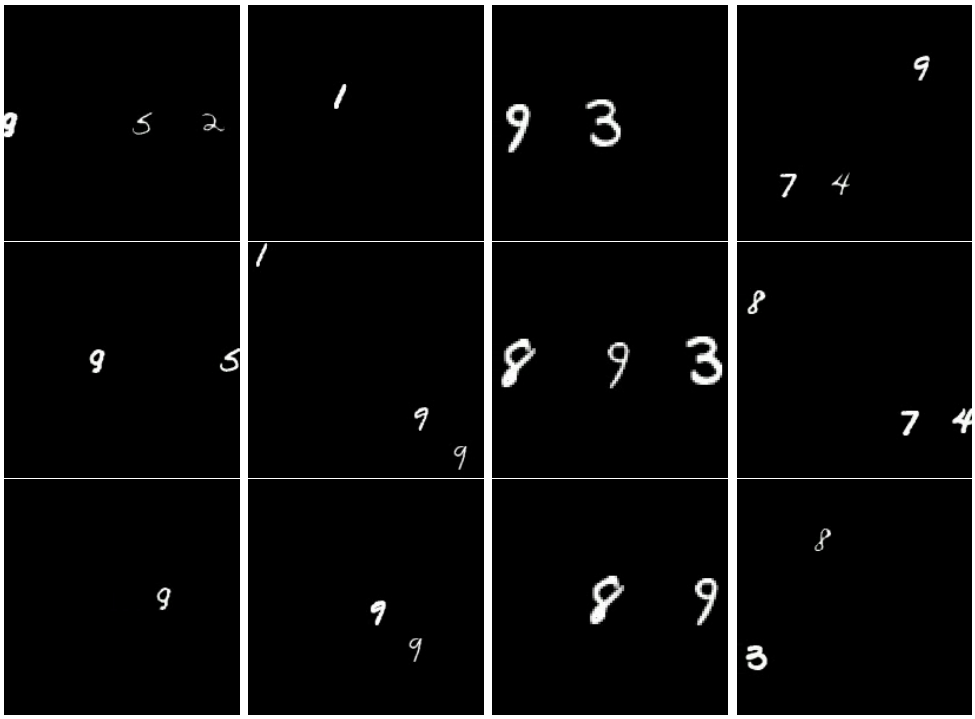
Remark that the *DeepMotionCLS* is not lifted to CTC version. This is due to the fact that the CTC requires the output sequence to be twice as long as the number of objects to be segmented. As the *DeepMotionCLS* processes the input sequence second by second, the maximum number of objects to be found would be of 10, which is restrictive in practice.

The *DeepMotionCLF-CTC* processes the input sequence frame by frame, then, for each frame, it outputs whether or not a new object appears. Using the CTC allows to get rid of the long term dependency problem inherent to the *DeepMotionCLF* approach. As the network only has to assess if an object has appeared, it only requires short term memory. However, this architecture inherits the limitations in parallelization of *DeepMotionCLF*.

As the CTC breaks the long term dependencies in the case of flow rate estimation, using an architecture only based on 3D convolutions is a promising lead. The convolutions can be used to gather information spatially and temporally, while reducing the dimensionality of the input data. This reduction, coupled with the high parallelization capabilities, allows for an extremely fast prediction speed. We experiment such approach in the *DeepMotion3D-CTC* architecture. Although unconventional in the sense that the CTC was originally designed to be used with RNNs, we believe that this approach is very promising given this peculiar problem.

4.2.4 A synthetic dataset: Moving Digits

To evaluate the proposed architectures for the estimation of the flow rate, we introduce a new evaluation dataset: Moving Digits. The main goal is to provide with a framework to test the behavior of the networks in various scenarios that will mimic real life conditions of road tunnel recordings. In particular, we target the three following issues: change of camera angle, change of object scale and change of number of lanes.



(a) Digits moving from left to right

(b) Digits moving in diagonal

(c) Digits with a different scale

(d) Two flows of digits (left to right)

Figure 4.8 – Example of simulated frames by our generator. The frames are shown in chronological order on each column. The first column represents digits moving from left to right. The second column shows digits moving with a different orientation, from bottom right to top left. The third column shows digits that are upscaled by a factor of two. The last column illustrates digits moving on two separate flows.

The dataset we elaborate is based on MNIST [LeCun and Cortes, 2010]. It is composed of randomly selected digits crossing a black screen in a given direction. We build a generator so as to control the various target parameters during generation. The generator takes in a configuration file and a number of videos to generate and outputs three sets: train, validation and test. The configuration file contains the description for each flow of digits to appear on screen:

- coordinates on the screen of the start and exit points,
- a scale factor,
- a speed value,
- the maximum number of digits in the stream,
- the generation frequency ratio.

The trajectory between the entry and the exit point is linear. The scale factor is unique for a given flux but can change across flux (although in the experiments we keep it constant). The speed value represents the number of frames the object will take to go from the entry point to the exit one. Finally, the generation frequency ratio is the probability of an object appearing at the start of the flux at each frame. Such generation being conditioned on the defined maximum number of digits. Note that, we randomly apply dilation and erosion to the digits throughout their displacement to add some noise and avoid too simplistic dataset. Example of generated frames are shown in Figure 4.8.

We generate multiple datasets to test the three degrees of freedom. For the change of camera orientation, we generate datasets with one flux of digit that always cross the center of the video frames. We consider the following angle values {0, 45, 90, 135, 180, 225, 270, 315}, w.r.t the abscissa 0° being a horizontal flux from left to right. Flows ending in the top of the frames (45°, 90°, 135°) can be considered to mimic tunnel cameras looking at the rear of vehicles ("going"), the flows ending at the bottom (225°, 270°, 315°) mimic cameras looking at the front of vehicles ("coming") and the remaining orientations (0°, 180°) are in-between position neither "coming" nor "going". Examples for 0° and 135° are respectively given in Figure 4.8a and Figure 4.8b. For scale, we generate three datasets with an angle of 0° and the scale ratio varying in {1, 2, 4}. Example frames for an upscale of $\times 2$ are provided in Figure 4.8c. Finally, we experiment with two flows (resp. above and below x-axis) for the angle fixed at 0° and a scale at 1 (see Figure 4.8d). For all those setups, we set the update ratio to 0.01, the maximum number of digits to 20 and the speed to 120 frames. Videos are 20 seconds long, at a frame rate of 25 FPS and 200x200 pixels in width and height. For each explored parameter, we simulate three sets (train, validation and test) with respectively 10,000, 2,000 and 5,000 videos.

4.2.5 Experiments

Experiments are conducted to assess the quality of the proposed architectures, as well as the influence of the angle, scale and number of flow on the accuracy of the flow rate prediction. We use the generated Moving Digit dataset for training and evaluation. As this works is aimed towards a setting where available data is scarce and not available for all the camera configurations (end application is Actemium's tunnel dataset), we mainly study the generalization capabilities of the networks. The results are evaluated on the generated datasets described above. We use the Mean Absolute Error (MAE) to measure the accuracy of the networks. In case of multiple flows of objects, the results per flow are averaged. In the rest of the chapter, we call source domain, the dataset setting used to train the networks and target domain the ones the networks are tested upon.

Implementation details The networks are trained on a single GPU. We use the Mean Squared Error (MSE) as training loss so as to reduce large prediction errors. Given that we do not know how changes in the recorded videos can affect the MV frames, we do not apply data-augmentation. We use an Adam optimizer [Kingma and Ba, 2015] and set the batch size to 32 and do not apply weight decay. Within each setting, the networks are trained once (the procedure is not repeated), so as to demonstrate the ease of training of each architecture.

The proposed architectures based on regression (*DeepMotionCLF*, *DeepMotionCLS*, *DeepMotion3D*) are detailed in Table 4.1 (c.f Figure 4.6 for illustration). As the MV frames do not contain texture information, we do not use as many convolution filters as we would with classical RGB frames. For the *DeepMotionCLF* architecture, we use two 2D convolutions with stride 2 so as to reduce the spatial dimension of the input data, then, we apply a ConvLSTM layer and finally use two consecutive dense layers. Similarly, for the *DeepMotionCLS* network, we use two 3D convolutions, with stride 2, we further add an average pooling layer to compact the information on the time axis, then, we also apply the ConvLSTM layer followed by dense layers. Finally, for the *DeepMotion3D* architecture, we use two 3D convolutions followed by an average pooling layer and directly use two dense layers for prediction.

DeepMotionCLF		*DeepMotionCLS*		*DeepMotion3D*	
Layers	Output Size	Layers	Output Size	Layers	Output Size
input layer	$500 \times 13 \times 13 \times 2$	input layer	$20 \times 25 \times 13 \times 13 \times 2$	input layer	$500 \times 13 \times 13 \times 2$
$[3 \times 3, 32$ (s2)]	$500 \times 6 \times 6 \times 32$	$[3 \times 3 \times 3, 64$ (s2)]	$20 \times 12 \times 6 \times 6 \times 64$	$[3 \times 3 \times 3, 64$ (s2)]	$249 \times 6 \times 6 \times 64$
$[3 \times 3, 32$ (s2)]	$500 \times 2 \times 2 \times 32$	$[3 \times 3 \times 3, 64$ (s2)]	$20 \times 5 \times 2 \times 2 \times 64$	$[3 \times 3 \times 3, 64$ (s2)]	$124 \times 2 \times 2 \times 64$
CL-$[2 \times 2, 32]$	$1 \times 1 \times 1 \times 32$	AvgPool-$[5 \times 1 \times 1]$	$20 \times 1 \times 2 \times 2 \times 64$	AvgPool-$[2 \times 2 \times 2]$	$62 \times 1 \times 1 \times 64$
-		CL-$[2 \times 2, 64]$	$1 \times 1 \times 1 \times 1 \times 64$	Reshape	$1 \times 1 \times 1 \times 3986$
FC-64	$1 \times 1 \times 1 \times 64$	FC-64	$1 \times 1 \times 1 \times 1 \times 64$	FC-64	$1 \times 1 \times 1 \times 64$
FC-1	prediction	FC-1	prediction	FC-1	prediction

Table 4.1 – Proposed regression networks. *DeepMotionCLF* process the inputs frame per frame, *DeepMotionCLS* second per second and *DeepMotion3D* ignores the sequential aspect of the data. For the ConvLSTM-based networks, only the last time step is returned.

As for the two CTC-based architectures (*DeepMotionCLF-CTC*, *DeepMotion3D-CTC*), we detail them in Table 4.2 (c.f Figure 4.7 for illustration). Both the *DeepMotionCLF-CTC* and *DeepMotion3D-CTC* use the exact same processing pipeline as their non CTC counterpart (*DeepMotionCLF* and *DeepMotion3D*). However, the ending dense layers are applied at each time step output rather than at only the last one and they are duplicated so as to match the number of flows to be predicted on the videos. Note that such network only allows prediction on videos with similar number of flows. Indeed, in case of dissimilar number of flows, if more flows are to be predicted, outputs will be missing and, if fewer flows are to be predicted, there is no way to know which outputs will be used by the network.

Finally, we evaluate the number of FPS that can be processed for each of the networks. We use a NVIDIA GTX 1080, set the batch size to 8 and run 1000 predictions. As the proposed architectures are extremely fast due to their reduced size when compared with classical RGB-based networks, we preload the datapoints into memory to avoid the data input bottleneck. The final FPS value is the average over the total number of predictions.

Baselines We start by experimenting the estimation of q from the compress MV frames by evaluating the quality of the methods when the training and the testing sets have similar constraints. We run one training for each target orientation, scale and number of flow and report the results in Table 4.3. Regarding the regression networks, they almost always provide with accurate predictions. Only the *DeepMotionCLF* architecture seems to be a bit more unstable, with 3 performances above 1 in MAE. Conversely, the CTC-based networks have much more difficulties finding a satisfactory minimum, with 8 values of MAE above 2.8. In such cases, the trained networks either output a constant value of 0 or largely underestimate

DeepMotionCLF-CTC		*DeepMotion3D-CTC*	
Layers	Output Size	Layers	Output Size
input layer	$500 \times 13 \times 13 \times 2$	input layer	$500 \times 13 \times 13 \times 2$
$[3 \times 3, 32$ (s2)]	$500 \times 6 \times 6 \times 32$	$[3 \times 3, 32$ (s2)]	$249 \times 6 \times 6 \times 32$
$[3 \times 3, 32$ (s2)]	$500 \times 2 \times 2 \times 32$	$[3 \times 3, 32$ (s2)]	$124 \times 2 \times 2 \times 32$
CL-$[2 \times 2, 32]$	$500 \times 1 \times 1 \times 32$	AvgPool-$[2 \times 2 \times 2]$	$62 \times 1 \times 1 \times 32$
[FC-64] $\times n_{flow}$	$500 \times 1 \times 1 \times 64$	[FC-64] $\times n_{flow}$	$62 \times 1 \times 1 \times 64$
[FC-2] $\times n_{flow}$	prediction	[FC-2] $\times n_{flow}$	prediction

Table 4.2 – Proposed CTC-based networks. The dense layer at the end of each network is applied once per time step. One prediction is output per flow, with one unique dense layer per flow.

the number of objects, hinting towards networks stuck in bad local minima. However, it is important to note that, for the *DeepMotion3D-CTC* network, when the training manages to find a proper optimum, it provides with the smallest prediction error.

	Orientation								Scales			Number of flows	
	0°	45°	90°	135°	180°	225°	270°	315°	×1	×2	×4	1 flow	2 flows
Regression based:													
DeepMotionCLF	0.13	0.17	0.24	0.18	1.47	**0.18**	0.17	0.19	0.13	0.19	1.42	0.13	1.03
DeepMotionCLS	0.10	0.19	0.22	0.21	0.22	0.19	0.10	**0.15**	0.10	0.17	0.28	0.10	0.13
DeepMotion3D	0.19	0.20	0.25	0.19	0.23	0.20	0.19	0.16	0.19	0.25	0.37	0.19	0.17
CTC based:													
DeepMotionCLF-CTC	0.24	0.21	3.40	3.32	**0.16**	0.33	3.28	3.37	0.24	0.25	3.37	0.24	0.26
DeepMotion3D-CTC	**0.04**	**0.14**	**0.06**	**0.08**	2.89	3.40	**0.02**	3.37	**0.04**	**0.05**	**0.17**	**0.04**	**0.08**

Table 4.3 – q value prediction when training and testing on generated datasets with similar settings. Higher errors are highlighted using a more vivid orange and the best accuracy for each column is in bold.

Effects of the direction of the flows We detail the impact of the change of flow orientation (respectively the first and second column of Figure 4.8). The proposed networks are trained on each orientation (from 0° to 315°) and, for each training, evaluated on the remaining orientations. In Table 4.4 we report the results when training on the orientation 0°. Table 4.5 provides with results for the training on each orientation of the architecture *DeepMotion3D*. The full results for each network on each orientation are available in the appendices, Table C.1.

	Source	Target						
	0°	45°	90°	135°	180°	225°	270°	315°
Regression based:								
DeepMotionCLF	0.13	3.14	3.66	**1.45**	0.68	2.44	3.21	1.91
DeepMotionCLS	0.10	3.08	2.62	2.26	1.14	2.92	2.61	2.72
DeepMotion3D	0.19	**2.49**	2.55	2.74	**0.45**	**2.11**	**2.40**	2.29
CTC based:								
DeepMotionCLF-CTC	0.24	4.35	**2.03**	1.74	1.02	4.33	2.69	**1.46**
DeepMotion3D-CTC	**0.04**	4.37	4.38	4.30	0.87	4.39	4.26	4.36

Table 4.4 – q prediction results when training with angle 0 and testing on flows with angle >0°. Higher errors are highlighted using a more vivid orange. Overall the networks show poor generalization capabilities except for the opposite flow (180°). Furthermore, no method (with or without CTC) seems to outperform the others.

Overall, results from Table 4.4 show that none of the proposed architectures properly generalize when the camera angle are changed (which is a desirable property as data is scarce in practice). However, in the case of inputs with opposite angles (such as 0°/180°), the networks seem to generalize better (the MAE ranges from 0.45 to 1.14) than for the other angles. Furthermore, as shown in Figure 4.9, even with an MAE of 1.14, the network predictions are correctly aligned with the real outputs. However Table 4.5 suggests that opposite orientation does not always provide with accurate results. Moreover, if the network generalizes well from one orientation to another, the reverse does not hold. For instance, when trained on the source orientation 0°, *DeepMotion3D* correctly predicts on the orientation 180° (MAE of 0.45). In the way around (180° to 0°), the achieved MAE is only of 2.16 (see Table 4.5).

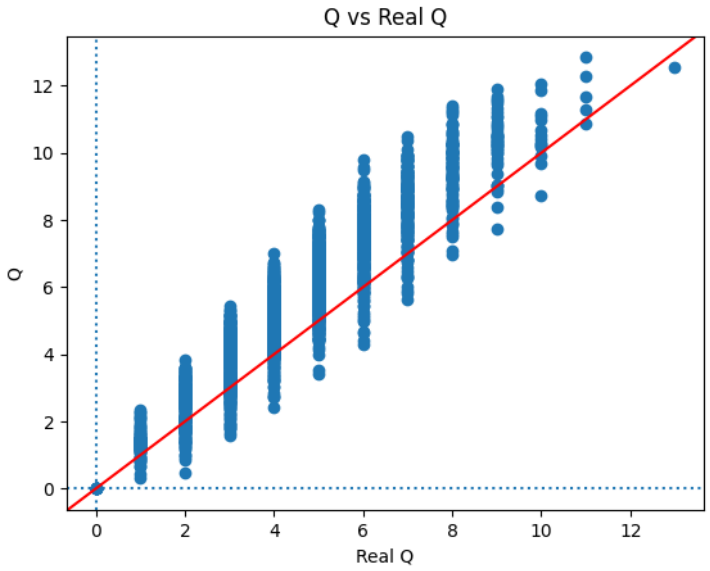


Figure 4.9 – Predicted q vs. real q values when training the network *DeepMotionCLS* on the flow orientation 0° and testing on the flow orientation 180°. The red line represents the function of equation $y = x$. Although the MAE precision is of 1.14, the network still provides with a correlation of 0.94 between the target and estimated values.

		Target Angle							
		0°	45°	90°	135°	180°	225°	270°	315°
Source Angle	0	**0.19**	2.49	2.55	2.74	*0.45**	2.11	2.40	2.29
	45	2.26	**0.20**	1.86	*1.13**	0.99	1.47	2.28	2.91
	90	*1.70**	*1.68**	**0.25**	2.01	1.34	2.13	*0.76**	2.41
	135	2.95	2.97	2.84	**0.19**	0.79	2.09	2.75	1.99
	180	2.16	2.60	2.27	2.22	**0.23**	1.95	2.21	2.64
	225	2.53	2.22	2.31	1.87	0.93	**0.20**	2.07	2.84
	270	2.42	2.61	*0.53**	2.20	2.16	2.35	**0.19**	*1.88**
	315	2.21	3.38	2.15	2.75	1.33	*1.09**	1.21	**0.16**

Table 4.5 – Results for the training of the DeepMotion3D architecture on each of the available orientations. Best results are shown in bold and second best in italic with a star. Higher errors are highlighted using a more vivid orange. As can be expected, the network performs best when tested on the same angle as at training stage.

We then aim to test the networks in case they are provided with data from all tested orientations but one to see how this helps the networks to better generalize. Hence, we train each of the networks in that setting and report the accuracy on the left-out camera's angle

in Table 4.6. The results led us to the following remarks. The *DeepMotionCLF-CTC* is hard to train with poor accuracy results. Overall, the CTC based networks yield higher errors. However, such result may be caused by the fact that the CTC networks only predict integer results. As such, for the CTC-based networks, each error is at least of order of 1, when regression networks may have smaller errors. For the other networks, we observe that they always generalize well to the unseen orientation. Such results strongly hint towards the fact that, if a large enough pool of videos is provided, the proposed regression-based architectures should be able to well generalize to videos from unseen cameras.

	Source	Target (All but one)							
	Avg	0°	45°	90°	135°	180°	225°	270°	315°
Regression based:									
DeepMotionCLF	**0.14** ± 0.04	0.86	0.67	0.34	**0.47**	0.59	**0.36**	**0.36**	0.91
DeepMotionCLS	0.16 ± 0.04	**0.55**	**0.59**	0.72	0.55	0.92	0.44	0.90	0.63
DeepMotion3D	0.18 ± 0.03	1.04	1.15	**0.33**	0.84	**0.40**	0.66	0.37	0.87
CTC based:									
DeepMotionCLF-CTC	1.92 ± 1.37	1.60	3.38	3.80	3.32	3.39	2.32	3.26	**0.39**
DeepMotion3D-CTC	0.48 ± 0.95	0.79	0.93	1.04	1.06	0.82	0.74	0.96	2.90

Table 4.6 – Prediction error when training on all orientations but one. Each target column is the one left out and tested upon. Source results are the average for all the source orientations. Best results according to each target angle are marked in bold font. Higher errors are highlighted using a more vivid orange.

Impact of the scaling factor We now aim to study the impact of objects' scale on the quality of the predictions. By this, we intend to evaluate the influence of the camera location according to the objects to be monitored (close to or far from the camera). To do so, we generate three datasets with a fixed angle (0°) and varying scale (×1, ×2 and ×4). The proposed networks are then trained on each dataset (corresponding to one scale) and evaluated on the two remaining ones. Results are reported in Table 4.7

	Source	Target		Source	Target		Source	Target	
	×1	×2	×4	×2	×1	×4	×4	×1	×2
Regression based:									
DeepMotionCLF	0.13	0.99	1.63	0.19	0.97	1.96	1.42	1.53	1.52
DeepMotionCLS	0.10	1.26	1.54	0.17	0.51	**0.61**	0.28	2.10	1.89
DeepMotion3D	0.19	**0.88**	**0.62**	0.25	0.73	0.93	0.37	1.85	1.88
CTC based:									
DeepMotionCLF-CTC	0.24	1.35	2.76	0.25	0.33	1.42	3.37	3.42	3.41
DeepMotion3D-CTC	**0.04**	1.85	2.26	**0.05**	**0.20**	1.82	**0.17**	**0.75**	**0.68**

Table 4.7 – Prediction results when training at multiple scale. Higher errors are highlighted using a more vivid orange. First panel is when training at scale ×1 and predicting at ×2 and ×4, second panel is training at scale ×2 and last at scale ×4.

Interestingly enough, CTC-based networks seem to achieve better performances. Especially, when trained on data of a given scale, they can correctly estimate the flow rate at smaller scales. However, we note that the *DeepMotionCLF-CTC* yields the worst results on the source data at scale ×4. Finally, regarding the regression-based network, no clear pattern emerges as to whether an architecture can generalize well to smaller or bigger objects in the videos. However, it is worth noticing that the predictions are usually highly correlated (> 0.8) with the real outputs, but with the flow rate values being under or over estimated. For instance, this is illustrated in Figure 4.10 of the plot when training the architecture *DeepMotionCLS* on scale ×4 and predicting on scale ×1.

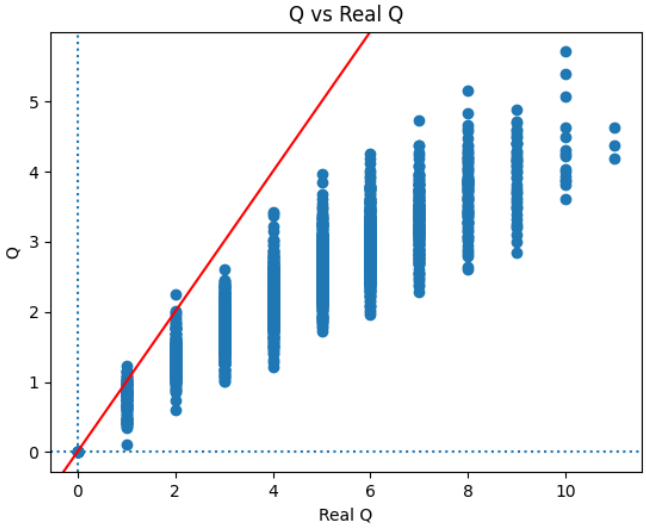
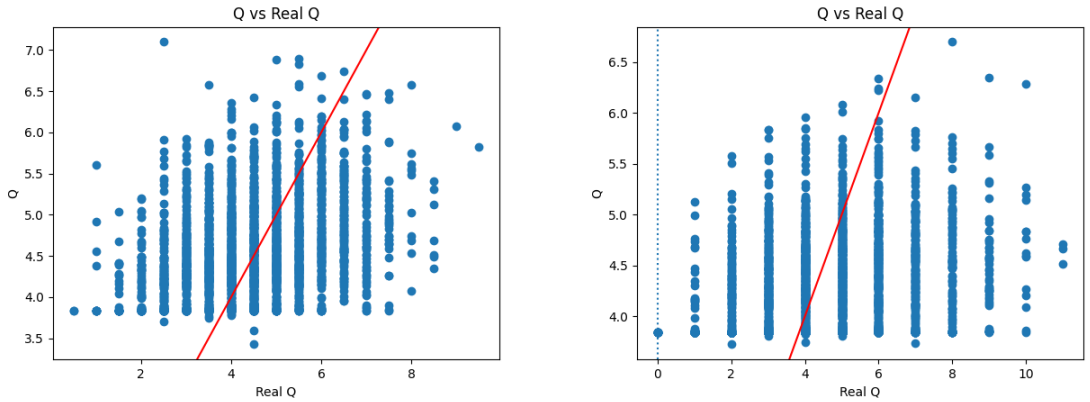


Figure 4.10 – Predicted q vs. real q , for the *DeepMotionCLS* architecture trained on the $\times 4$ scale and used for prediction on the $\times 1$ scale. While correlation between q and real q is high (0.91), the network grossly under-estimate the flow rate value

.

Impact of the number of flows We use two datasets, with constant scale and orientation ($\times 1$ and $0°$), and generate data for respectively 1 and 2 flows. Obtained results are reported in Table 4.8.



(a) Two flows test dataset.

(b) One flow test dataset.

Figure 4.11 – Predicted q vs. real q for *DeepMotionCLF* architecture for estimating the flow rate of two flows of digits. In both plots, the network fails to correctly correlate the predictions and real values. Expected output is shown in Figure 4.9.

First, *by design* the CTC based architectures cannot generalize to data with more/less flows than the one they were trained on. Indeed, as the number of flows is pre-set for the CTC models, they cannot predict on data with a different number of flows. Regarding the regression-based networks, we can see that none of them manages to correctly generalize to data with a different number of flows. Furthermore in the case of the *DeepMotionCLF*, according to Figure 4.11b, we see that while the MAE is somewhat low (1.03) when training on two flows, the network fails to correctly count the number of objects within the video streams. Such results support the previous conclusion that the *DeepMotionCLF* architecture is hard to train. These difficulties in training are likely to be explained by the challenging number of frames ($L = 500$). As the number of time steps increases, the network needs to

learn longer time dependencies.

	Source 1 flow	Target 2 flows	Source 2 flows	Target 1 flow
Regression based:				
DeepMotionCLF	0.13	3.97	1.03	**1.46**
DeepMotionCLS	0.10	3.51	0.13	4.09
DeepMotion3D	0.19	**1.91**	0.17	1.82
CTC based:				
DeepMotionCLF-CTC	0.24	-	0.26	-
DeepMotion3D-CTC	**0.04**	-	**0.08**	-

Table 4.8 – Prediction results when training with different number of flows. The CTC-based networks are not tested on the target sets as they do not allow it by design. Higher errors are highlighted using a more vivid orange.

Speed and memory comparison Finally, we study the speed and memory footprint of each of the proposed architectures. Their evaluations at that regard are reported in Table 4.9. As could be expected, the frame-based networks (*DeepMotionCLF* and *DeepMotionCLF-CTC*) use the most memory space on the GPU and are the slowest due to the sequential processing of each frame using a ConvLSTM layer, therefore preventing the parallelization of the networks. Still, these networks are able to process about 40 Datapoints Per Second (DPS) or about 20,000 FPS. Given Table 3.1, even these "worst" architectures are order of magnitude faster (×67.6 when considering a detection network with a FPS rate of 300) than any detection networks. Then, in increasing speed order, the *DeepMotionCLS* architecture (also based on ConvLSTM) is the slowest at approximately 580 DPS, the *DeepMotion3D-CTC* network comes second at 1,215 DPS and finally, the *DeepMotion3D* architecture is the fastest at 1,619 DPS or 809,567 FPS. Such results were to be expected as the *DeepMotion3D*-based architectures only use 3D convolutions and therefore can easily leverage the parallelization capabilities of the GPU. Furthermore, it is important to notice that the non frame-based architectures (*DeepMotionCLS*, *DeepMotion3D*, *DeepMotion3D-CTC*) use less than 500 Mebibyte (MiB) on the GPU, showing great capacities for deployment in embedded situations.

Network	Accuracy	GPU Memory (MiB)	DataPoints/sec	FPS
Regression based:				
DeepMotionCLF	0.13	3,555	41	20,275
DeepMotionCLS	0.10	487	582	290,934
DeepMotion3D	0.19	487	**1,619**	**809,567**
CTC based:				
DeepMotionCLF-CTC	0.24	3,565	40	19,808
DeepMotion3D-CTC	**0.06**	**475**	1,215	607,347

Table 4.9 – Speed comparison of the various architectures. Batch size is set to 8 for the inference. MiB stands for Mebibyte (1 Mi equals 1,048,576 and one Mb equals 1,000,000). All of the prediction networks provide with impressive processing speed.

4.2.6 Synthesis

All the proposed approaches can be used to correctly estimate traffic flow parameters. However, as expected from deep learning methods, the lack of data is a huge hindrance to the

generalization capabilities. More specifically, the networks do not manage to correctly generalize in a consistent manner when the source dataset is too restrained. Still, when training the networks with more data (orientation "all but one", experiment Table 4.6), they are able to correctly generalize to unseen setups. Furthermore, experiments tend to indicate that data with similar orientations are more easily generalized to. However, it is important to note that frame-based network *DeepMotionCLF-CTC* is hard to train by comparison with the other networks (c.f Table 4.6). Finally, both the CTC-based and non CTC-based approaches provide with similar accuracy levels in each experimental settings, validating both methods. Nonetheless, given that the CTC-based networks are a bit more unstable at training, the regression-based models are more recommended for industrial deployments.

Regarding prediction speed, while the frame-based networks are the slowest, they still provide with a significant advantage in FPS when compared to results on classical detection networks (c.f Table 3.1). Even if we were to apply the detection method proposed chapter 3 ($\approx$ 300 FPS), the slowest method presented here would be about 67 times faster. And, if we were to compare with the fastest method (*DeepMotion3D*), we get to an impressive result of more than 2,600 times faster while requiring very little memory. Given these results, the *DeepMotion3D* architecture seems to be the more promising one as it is the fastest while being on par with the other networks in terms of accuracy.

Overall, the results tend to indicate that using the compressed MV frames as input rather than the RGB frames is beneficial. All the tested architectures manage to reach similar level of accuracy while largely improving prediction speed when compared with RGB processing (accuracy comparison between RGB-based and MV-based networks is provided in chapter 5). Once presented with a correct solution to generalize to any unseen data in case of scare training datasets, compressed input based networks will become very interesting tools for low cost large scale deployment.

4.3 Domain Adaptation

In the previous section, we have seen that the proposed networks can learn to estimate the flow rate of moving objects from the compressed motion vector representation. However, in case of scarce data, the networks fail to generalize well. Such cases are bound to occur in industrial environments. For instance, in the cases of Paris' tunnels, due to technical constraints, it is not possible to get labelled data covering all the camera orientations, number of flows, etc. Therefore, we explore domain adaptation so as to adapt trained network to provide correct prediction on unseen and unlabeled data.

We redefine our problem in the light of domain adaptation. Given a set $S^s = \{\mathbf{x}_i^s, y_i^s\}$ of source domain cameras with labeled examples and a set $S^t = \{\mathbf{x}_i^t\}$ of target domain cameras without labels, we want to train a network on the source set S^s so as to output correct predictions on the target set S^t. Let $(\mathbf{x}^s, y^s) \sim p^s(\mathbf{x}, y)$ be the joint source distribution and $(\mathbf{x}^t, y^t) \sim p^t(\mathbf{x}, y)$, defined similarly, be the target distribution. The aim is to learn a prediction function $h(\mathbf{x}) = f \circ g(\mathbf{x})$, where $g : \mathcal{X} \to \mathcal{R}^d$ is a function that learns the feature representation $\mathbf{z}$ (embedding) from the raw input $\mathbf{x}$. Function f predicts the desired output y based on $\mathbf{z}$. Model h trained on S^s may not predict well on S^t, as in our application several variations may occur across cameras. Indeed, the changes in camera orientation angle or camera distance to the monitored objects likely induce changes in the statistical marginal distribution $p_{\mathbf{x}}^s(\mathbf{x})$ and $p_{\mathbf{x}}^t(\mathbf{x})$. Hence, so is for $p_{\mathbf{z}}^s(g(\mathbf{x}^s))$ and $p_{\mathbf{z}}^t(g(\mathbf{x}^t))$. Also p_y^s, the source output distribution, may shift from the target one because of the flow rate or the number of flows. Therefore, as $p_{\mathbf{z}}^s \neq p_{\mathbf{z}}^t$ and $p_y^s \neq p_y^t$, we seek to learn a function h based on the source labeled set S^s and the unlabeled target one S^t able to predict approximately on S^t by aligning the joint distributions. Specifically, we intend to match $p^s(\mathbf{z}^s, y^s)$ with $p^t(\mathbf{z}^t, y^t)$ so as to adapt the distribution of learned source and target embeddings $\mathbf{z}^s$, $\mathbf{z}^t$ and cope with the shift in output distributions. To do so, we rely on DeepJDOT framework [Damodaran

et al., 2018].

4.3.1 DeepJDOT

DeepJDOT is a domain adaptation method that aims to estimate h by aligning the source and target embedded representation of the data (resp. $g(\mathbf{x}^s)$ and $g(\mathbf{x}^t)$) as well as their labels through optimal transport. Let $\Pi(\mu^s, \mu^t)$ be the space of joint probability distributions with μ^s and μ^t respectively the source and target marginal distributions. Optimal transport [Peyré and Cuturi, 2020] aims to find a joint probability $\gamma \in \Pi(\mu^s, \mu^t)$ so as to minimize the displacement cost:

$$OT(\mu^s, \mu^t) = \inf_{\gamma \in \Pi(\mu^s, \mu^t)} \int_{\mathbb{R}^2} c(\mathbf{x}^s, \mathbf{x}^t) d\gamma(\mathbf{x}^s, \mathbf{x}^t) \tag{4.2}$$

where $c : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$ is the cost function giving the cost of moving $\mathbf{x}^s$ towards $\mathbf{x}^t$. An emblematic problem that optimal transports aim to solve is earth moving problem [Monge, 1781]. Given a pile of dirt, how to transport it so as to form a second pile of dirt shaped differently while limiting the transportation cost (see Figure 4.12). Note that Equation 4.2

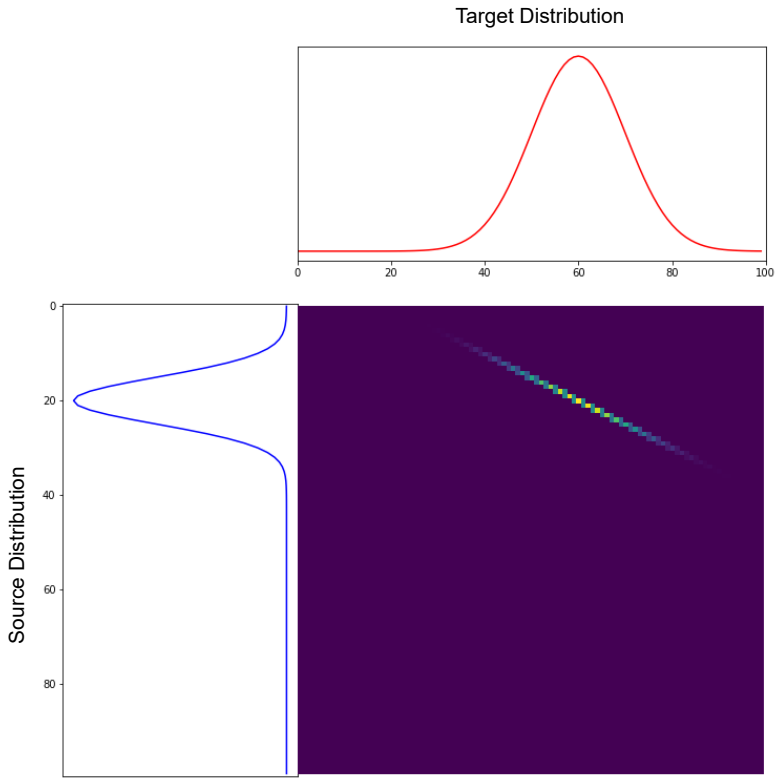


Figure 4.12 – Representation of the earth moving problem as well as the solution (example taken from POT library). The aim is to move the blue pile to match the red one. The blue matrix shows the related transport map γ.

provides a discrepancy measure between distributions μ^s and μ^t. DeepJDOT uses optimal transport to map the joint distributions $p^s(\mathbf{z}, y)$ and $p^t(\mathbf{z}, y)$, $\mathbf{z}$ being the embedding. More specifically, given a network trained on the source domain the DeepJDOT algorithm works as follows:

- embeddings are computed for a batch $B^s = \{(x_i^s, y_i^s)\}_{i=1}^{n_B}$ of source data and a batch $B^t = \{(x_i^t)\}_{i=1}^{n_B}$ of target data
- Using optimal transport, the coupling γ between the source and target embeddings is computed

- The coupling is used to provide with proxy labels for samples of the target batch
- Finally, the network is trained on B^s and B^t so as to minimize the distance between the coupled source and target embedding distributions, as well as the fitting error on source domain data.

More formally, let $h = f \circ g$, where $g : \mathbf{x} \to \mathbf{z}$ is the embedding function and $f : \mathbf{z} \to y$ is the prediction function, then the aim is to minimize the following objective:

$$\min_{\gamma \in \Pi(\mu^s, \mu^t), f, g} \sum_i \sum_j \gamma_{ij} d(g(\mathbf{x}_i^s), y_i^s; g(\mathbf{x}_j^t), h(\mathbf{x}_j^t)) \tag{4.3}$$

where $d(g(\mathbf{x}_i^s), y_i^s; g(\mathbf{x}_j^t), h(\mathbf{x}_j^t)) = \alpha \|g(\mathbf{x}_i^s) - g(\mathbf{x}_j^t)\|^2 + \lambda \mathcal{L}(y_i^s, h(\mathbf{x}_j^t))$ is the generated cost. Parameters α and λ control the trade of embedding distance/loss importance. As the efficiency of the function h on the source domain is to be preserved, one rather aims at:

$$\min_{\gamma, f, g} \frac{1}{n^s} \sum_i \mathcal{L}(y_i^s, h(x_i^s)) + \sum_i \sum_j \gamma_{ij} d(g(\mathbf{x}_i^s), y_i^s; g(\mathbf{x}_j^t), h(\mathbf{x}_j^t)). \tag{4.4}$$

In our case, this means that we must design a network such that the embedding $\mathbf{z}$ represents information for the whole sequence of MV frames $\mathbf{x}$ (c.f Equation 4.1). For the regression-based networks, we select the last vector of length 64 as embedding representation. For the CTC networks, there is no global representation as the optimization procedure considers the output at each time step. Hence, we match each time step's embedding of the source sequences to a time step's embedding in the target sequences *globally*. Such coupling allows to minimize the discrepancy between embedding distributions, however, it does not allow to apply the CTC loss to the target proxy outputs. Indeed, in its original formulation, DeepJDOT optimizes the target output using proxy values. In the case of the CTC, the loss needs the final output sequence (for instance "oooo" in case of 4 objects) to run the optimization. However, as we match source and target embeddings globally (i.e disregarding the sequences), each target sequence's label after coupling can not be inferred. Therefore, in case of the CTC, for the target domain, we use the source probability output as proxy for each target output and apply the squared L2 distance (euclidean distance) as loss.

4.3.2 Experiments

Experiments are conducted to assess the ability of the domain adaptation to cope with the variations on the cameras, namely the angle, scale and number of flows. We use the same experimental setups as previously. However, to compute meaningful distance between distributions in the DeepJDOT framework, we increase the batch size to 50. We keep all the other parameters as proposed in the DeepJDOT article. We report the previously obtained results and compare them with the ones we obtain after domain adaptation. It is important to notice that due to GPU memory constraints, we were not able to carry domain adaptation for the frame based networks, namely *DeepMotionCLF* and *DeepMotionCLF-CTC*.

Impact of the orientation of the flows We use as source the networks trained on the camera with orientation angle of 0° and adapt to the other configurations. We report the results in Table 4.10. Prediction error on the source domain after adaptation is the average of each prediction error obtained on the source test set after adapting on each of the target cameras. Overall, for the non CTC models, the adaptation works perfectly, does not increase the error on the source domain and reduces it on the target domains. The results for the CTC-based network are more mitigated. Although the adapted networks reduce the prediction error for all the target sets, we can notice that the error is still high on some of the target sets (for instance 1.67 at 225°). Such behavior can probably be explained by the fact that, on the target domain, the CTC loss can not be applied due to the matching of each time step separately through optimal transport.

	Source		Target													
	0		45		90		135		180		225		270		315	
Regression based:																
DeepMotionCLF	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
DeepMotionCLS	0.10	0.10	3.08	0.55	2.62	**0.61**	**2.26**	0.56	1.14	0.56	2.92	0.47	2.61	0.47	2.72	**0.45**
DeepMotion3D	0.19	0.19	**2.49**	**0.41**	**2.55**	0.68	2.74	**0.54**	**0.45**	**0.43**	**2.11**	**0.38**	**2.40**	**0.46**	**2.29**	0.53
CTC based:																
DeepMotionCLF-CTC	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
DeepMotion3D-CTC	**0.04**	**0.04**	4.37	0.77	4.38	1.36	4.30	1.15	0.87	0.80	4.39	1.67	4.26	0.68	4.36	1.13

Table 4.10 – Results for domain adaptation from orientation of 0° towards all the other orientations. Grayed cells denote the accuracy before the adaptation. A bold font indicates the best results. A green cell shows the case where adaptation improved the accuracy.

Impact of the scaling factor We perform the adaptation from each available scale to the remaining ones and report the results in Table 4.11. Overall we observe a similar behavior as for the orientation of the flows. The non CTC methods perform best with impressive improvements on the target domain while, at most, increasing the error on the source set by 0.01. Although improving the results, the adapted CTC method still presents a high level of error, as before.

	Source		Target				Source		Target				Source		Target			
	×1		×2		×4		×2		×1		×4		×4		×1		×2	
Regression based:																		
DeepMotionCLF	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DeepMotionCLS	0.10	**0.10**	1.26	0.60	1.54	0.58	0.17	0.17	0.51	0.36	**0.61**	**0.58**	0.28	0.29	2.10	**0.59**	1.89	**0.60**
DeepMotion3D	0.19	0.19	**0.88**	**0.55**	**0.62**	**0.54**	0.25	0.24	0.73	0.42	0.93	0.59	0.37	0.36	1.85	0.66	1.88	0.70
CTC based:																		
DeepMotionCLF-CTC	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DeepMotion3D-CTC	**0.04**	0.19	1.85	1.71	2.26	0.59	**0.05**	**0.06**	**0.20**	**0.23**	1.82	1.42	**0.17**	**0.18**	**0.75**	0.67	**0.68**	0.74

Table 4.11 – Domain adaptation results for varying scales. Grayed area are the accuracies before adaptation. A green cell shows the case where adaptation improved the accuracy and a red cell where it decreased the accuracy.

	Source		Target		Source		Target	
	1 flow		2 flows		2 flows		1 flow	
Regression based:								
DeepMotionCLF	-	-	-	-	-	-	-	-
DeepMotionCLS	**0.10**	**0.09**	3.51	0.96	**0.13**	**0.13**	4.09	0.61
DeepMotion3D	0.19	0.19	**1.91**	**0.46**	0.17	0.17	**1.82**	**0.57**
CTC based:								
DeepMotionCLF-CTC	-	-	-	-	-	-	-	-
DeepMotion3D-CTC	-	-	-	-	-	-	-	-

Table 4.12 – Domain adaptation results when the number of flows is changing. Grayed area are the accuracies before adaptation. The *DeepMotion3D-CTC* is not adapted as the CTC representation does not allow it. A green cell shows the case where adaptation improved the accuracy and a red cell where it decreased the accuracy.

Impact of the number of flows We study the adaptation from one flow to two flows and reversely. As the CTC networks are *by design* not suited for such operation, the domain adaptation is restricted to regression based architectures. The results are shown in Table 4.12. Here again, the adaptation improves the prediction accuracy in all investigated settings.

Synthesis Overall, domain adaptation shows promising results. Whether it is for changes of orientation, scale or number of flows, the accuracy almost always improves. And, when it decreases, it only does so by a few hundredth. It is important to note that the *DeepMotion3D-CTC* architecture suffers from the worse adapted results. This probably links to the fact

that the adaptation breaks the sequence assumption, preventing from using the CTC loss on the target domain. Finally, few limitations are to be put forward. First, the frame-based networks, *DeepMotionCLF* and *DeepMotionCLF-CTC*, could not be adapted due to their size. As DeepJDOT requires large batches of source *and* target samples, it is hardly applicable to networks that require large chunks of memory during training. Second, due to the current design of the CTC-based networks, they cannot be adapted towards dataset with changing number of flows. While these limitations do not jeopardize the adaptation as a whole, they largely hinder the deployment of CTC-based architectures.

4.4 Conclusion

In this chapter, we have studied the estimation of object flow rate from MPEG4 part-2 compressed videos. We propose to use the compressed MV frames to replace the cumbersome RGB inputs and design several architectures to process them. In particular, we devise three models based on regression (*DeepMotionCLF*, *DeepMotionCLS* and *DeepMotion3D*) and two based on the CTC (*DeepMotionCLF-CTC*, *DeepMotion3D-CTC*). To test the proposed models, we introduce a simulation dataset: Moving Digit. Using this dataset, we have shown that the proposed solutions generalize well to unseen settings, when provided with enough data. However, our empirical findings reveal that this is not the case when training with limited number of examples. We devise a solution based on domain adaptation, in particular DeepJDOT. Using this framework, we show that it is possible to adapt trained networks to unseen data. Overall, using the MV frames for flow rate estimation is a very promising solution as *all* the proposed networks can be trained to reach an impressive accuracy on any of the tested dataset.

Regarding the speed of the proposed models, we highlight a large improvement of the frame processing speed over classical detectors. Moreover, the non frame-based architectures (*DeepMotionCLS*, *DeepMotion3D* and *DeepMotion3D-CTC*) require little memory, hence, perfectly fitting industrial constraints. If properly used, these methods can be of a huge support for tunnel surveillance while limiting deployment costs. We discuss in the next chapter application of the proposed methods to real data.

Although we provide models for flow rate estimation based on compressed data representation, we only exploit a fraction of the compressed information. For instance, we have overlooked the usage of the residual information. As the moving objects are easily detected in the residual matrices, it could be interesting to include such information to help the network filter out noisy motion vectors. More specifically, as residual matrices are compressed in the DCT domain similarly to JPEG images, and as a large part of the frequency coefficients can be discarded without impeding detection accuracy in JPEG images (c.f previous chapter), one could probably couple the MV frames with trimmed residual matrices to keep the computation costs low. Finally, the newly presented CTC-based methods, while interesting, are still faced with multiple issues. Adaptation, in the presented setting, does not allow to use the CTC loss on the target outputs. Using other algorithms, such as the Dynamic Time Wrapping (DTW), to match the whole sequence as one might be beneficial. Another challenge faced by the CTC-based networks is the lack of flexibility regarding the number of outputs. Recent work [Carion et al., 2020] have shown that transformers can be used for prediction when the number of outputs varies. It would be interesting to see if this architecture can be used to provide with more resilient CTC-based networks.

Chapter 5

Vehicle Counting: A Real Case Application

> “ *The plural of the word anecdote is not data.* ”
>
> Kenneth Kernaghan and P. K. Kuruvilla

Contents

In the previous chapter, we have presented deep architectures to estimate the flow rate of moving objects on a fixed background using Motion Vector (MV) of compressed videos. The proposed models are trainable in an end-to-end manner and amenable to domain adaptation. However, we have only tested the proposed methods on generated data. In this chapter, we are interested in their application to real-world data.

While the literature for traffic flow estimation is abundant, the related datasets are scarce. A large part of works introduce their own data, and, to the best of our knowledge, no dataset serves as reference. One of the most used for evaluation is the NVIDIA AI CITY challenge dataset [Naphade et al., 2017, 2018, 2019, 2020]. However, this dataset is subject to strict licensing that prevents anyone but the challenge participants to use it. This absence of reference dataset is due to both annotation difficulty and the fact that traffic videos are sensitive data. First, without access to induction loops, the annotation requires to detect each vehicle, to track it throughout the video frames and to extrapolate the traffic flow parameters from visual cues, such as lane marking. Given this process, annotating hours of traffic videos reveals a daunting task. Second, as the recorded videos often come from safety cameras, there is a requirement to both anonymize the vehicles and hide the locations of the cameras. These combined difficulties greatly limit the public release of datasets and, therefore, of new methods.

(a) Light saturation from braking vehicles. (b) Light saturation from headlights. (c) Truck occlusion (front). (d) Truck occlusion (back).

Figure 5.1 – Examples of noise present in the videos recorded onsite. Apart from dirt on the cameras' lens due to pollution, two main types of noise are present: illuminations (a and b) and occlusions (c and d).

Although we have shown in chapter 4, on a synthetic dataset, that deep models using MV frames as input can correctly estimate the flow rate, how such networks perform on real data remains an open question. Indeed, real tunnel images we are interested in suffer from various noises such as illuminations or occlusions (c.f Figure 5.1). In order to prove the efficiency of those deep models in real-life conditions, we collected and annotated our own dataset. Actemium has access to video recordings of road tunnels coupled with induction loop readings. This greatly simplifies the annotation process. Moreover, as the recorded data is of low visual quality, anonymization is made easier as the only information to be hidden is the incrusted camera tag (see the black square at the bottom left of images in Figure 5.1).

We present Actemium dataset and show interesting properties related to traffic flow theory. Then, we test our proposed method on this new dataset and demonstrate impressive speed-up gains while maintaining and even improving the accuracy when compared with a classical detection-tracking-estimation model. Relying on the methodological propositions from chapter 4 we also study domain adaptation for cases where data are scarce. We show limitations with the naive adaptation approach and explore a solution to improve estimation

accuracy on unseen cameras.

The remainder of the chapter is divided as follow. We start by introducing the basic of traffic flow theory, the Actemium dataset and interestingly relate the two. We conduct experiments on this new dataset, using the deep architectures (based on MV frames) presented in chapter 4. Furthermore, so as to provide a comparison baseline, we also propose with a RGB-based detection-tracking-estimation solution. Then, we develop on domain adaptation and show that the naive approach on real data leads to unstable network accuracy. Finally, we discuss on these limitations, relate them to discrepancies in the output distributions and empirically validate our hypothesis using oracle information.

5.1 Traffic flow theory and dataset

Traffic surveillance cameras record roads subjected to the phenomenons described by traffic flow theory. As such, it is likely that properties of traffic flow theory are to be tightly intertwined with the recorded data properties. In this section, we introduce the core principle of traffic flow theory, we detail the collected dataset and relate the two.

5.1.1 Definition of the usual flow measurements variables

The theory of traffic flow can be divided into two groups: microscopic and macroscopic analysis. Where the former describes traffic flow by considering each vehicle separately, the latter only considers the general state of the flow. As our end goal is to analyze traffic from road videos, we are only interested in the mathematical formulations of the microscopic approach[1]. In the real life setting, microscopic measurements are usually collected using induction loops. Hence, one may compute and/or estimate the main flow parameters which are: the flow rate q, the occupancy o, the velocity v[2] and the density ρ.

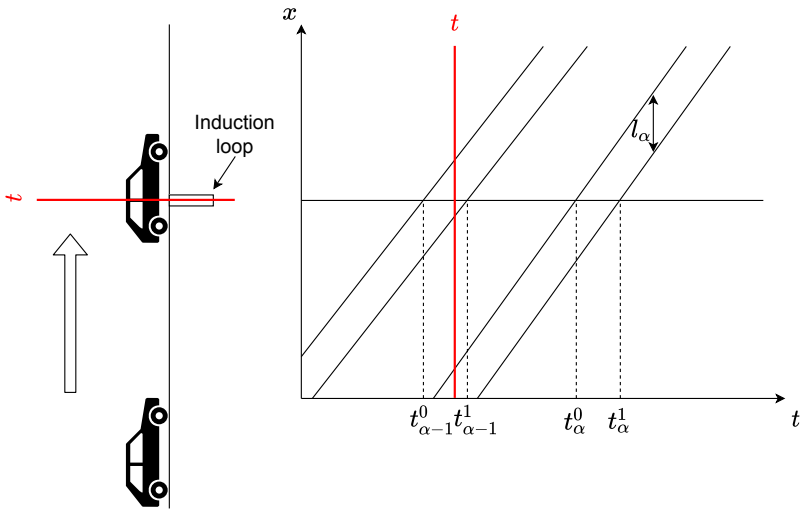


Figure 5.2 – Representation of the microscopic variables used to estimate the traffic flow parameters.

Let t^0_α and t^1_α be the time at which the front (resp. rear) of vehicle α crosses the induction loop (vehicle counting sensor) and l_α its length (c.f Figure 5.2). Then, the traffic flow q is defined as:

$$q(x,t) = \Delta N, \tag{5.1}$$

[1] All the information detailed in this section, as well as the figures, were taken and adapted from Immers and Logghe [2002].

[2] The velocity can only be computed with dual induction loops, as they allow to get a measurement of the vehicle length. Otherwise, only an estimate is available.

where ΔN is the number of vehicles that have passed the cross-section at location x within a time interval $[t, t+\Delta t]$ (Δt usually varies between 20 seconds and 5 minutes depending on the application). The occupancy o, which represents the average time during which a vehicle was over the induction loop, is defined as:

$$o(x,t) = \frac{1}{\Delta t} \sum_{\alpha=\alpha_0}^{\alpha_0+\Delta N-1} (t_\alpha^1 - t_\alpha^0). \tag{5.2}$$

The average velocity is defined as:

$$v(x,t) = \frac{1}{\frac{1}{m}\sum_{\alpha=\alpha_0}^{\alpha_0+m-1} \frac{1}{v_\alpha}}, \tag{5.3}$$

where v_α is the speed of vehicle α. It is to be noted that the harmonic mean is used rather than the arithmetic mean as the speed is measured over a fixed distance rather than a fixed period of time. In case of single loop detectors, v_α can be estimated using an approximated length $\bar{l}$ for all vehicles:

$$v_\alpha = \frac{\bar{l}}{t_\alpha^0 - t_\alpha^1}. \tag{5.4}$$

Finally, while q, o and v can be directly measured from induction loops, the density can only be estimated from the computed q and v as:

$$\rho = \frac{q}{v}. \tag{5.5}$$

Equation 5.5 is fundamental to traffic flow theory and leads to the diagrams presented in Figure 5.3. In the diagrams, v_f is the maximum velocity (or velocity at free flow), q_c, v_c and ρ_c are respectively the flow rate, velocity and density at the road maximum capacity and ρ_j is the density when the traffic is jammed. These diagrams are called the fundamental diagrams of traffic flow theory. It is important to note, as we later show, that they correlate interestingly with real data.

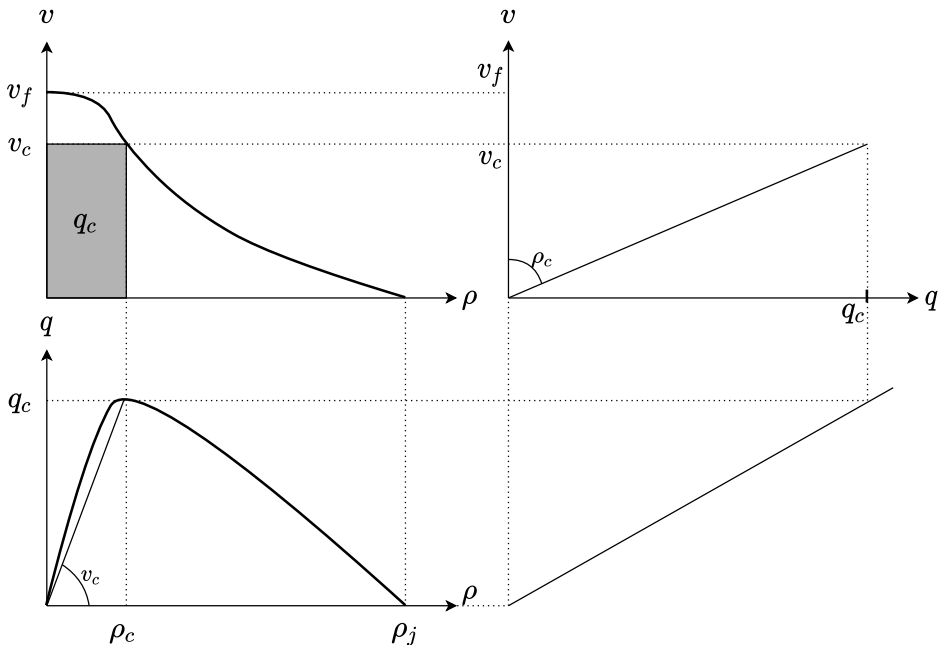


Figure 5.3 – Sketch of the fundamental diagrams of traffic flow theory. The diagram are road dependent and therefore, the shape of the curve may change depending on its associated road.

5.1.2 Actemium's Tunnel Video Dataset

Data collection for traffic flow estimation is a complex and daunting work. Leveraging actemium's access to both video recordings and induction loop readings, we annotate our own dataset fitting the task at hand. Data annotation remains a complex challenge due to various industrial limitations inducing approximations in the annotated data. Below, we introduce an overall comprehension of our annotation challenges and related limitations and then detail the collected data.

Data collection and annotation

We aim to create a dataset coupling video recordings and traffic flow information. Tunnel videos are recorded and encoded using the MPEG4 part-2 compression format by onsite coders. They are recorded at about 25 Frames Per Second (FPS) and the timestamp is embedded within the frames. The traffic flow is recorded from induction loops and stored by Automatic Data Recording (ADR) stations in log files at fixed intervals of 20 seconds. Available data are the flow rate and the occupancy. We found the velocity data to be either missing or unreliable. These variables are computed over a 20 seconds window. As both video and traffic data come from separate sources, we need to synchronize them so as to annotate the videos with the corresponding traffic flow labels. However two main problems arise: the non-synchronization of the time clocks and inconsistencies in the videos frame rate.

Coders and ADR stations time clocks are not synchronized, furthermore, each coder has its own clock. Therefore, the offset between each pair of coder/ADR station needs to be computed. Obviously the computation of such offset can only be done by matching the recorded flow rates with the visual video information. Luckily, the loop detectors compute the flow parameters for each lane, simplifying this matching. We build a script based on a detection-tracking-estimation method (similar to the one presented in subsection 5.2.1), which estimates the flow rate values per lane and find the best alignment with the recorded values. However, it is important to note that, as vehicles move at different speeds and can switch lane easily, video and flow rate can only be matched if the camera is looking at the induction loop. As such, only part of the cameras can be annotated and the number of effectively available cameras is largely reduced (at most 203 cameras out of 2,000).

The second issue is the inconsistency of the video frame rate. Although the theoretical frame rate is of 25 FPS, in practice, this is not the case. Overall, we found three main causes of frame shift by analyzing the data:

- Frames get lost during the data transfer over the intranet network, and, while this is visible on screen (visual artifacts), it cannot be seen from a pure data point of view.
- The coders sometimes encode only 24 frames for a second rather than 25 (probably for clock synchronization).
- Seconds might be skipped in the frame embedded time (i.e from 18:00:56 to 18:00:58, probably also for clock synchronization).

Hence, when associating video recordings with flow rate values, not only do we need to compute the offset between each ADR station and the associated camera, but we also must visually check that shifts do not occur in the stream. Such task is done in a semi-automated way using a script, which prompts the user for confirmation at given intervals. This methodology possibly implies a slight shift in the data, and therefore we choose to associate videos of 21 seconds with the recorded traffic flow measurements so as to ensure that each video encompasses the whole measurement period of flow parameters. Consequently, annotations might be noisy as more vehicles than the ones accounted for in the measurements may be visible on screen. In the end, given all the limitations, we were able to annotate data from 5 cameras (c.f Figure 5.4).

(a) E72.433F$_3$ (b) *E72.722Y*$_2$ (c) *E73.268Y*$_4$ (d) E73.332A$_4$ (e) E73.531M$_3$

Figure 5.4 – Screenshots of the recorded cameras. Each camera is identified by a tag. The index indicates the number of lanes. Tags in italic correspond to the cameras looking at the front of the vehicles.

Data analysis

We now detail the specificities of the collected data. The recordings were carried over 2 days at various hours. The timeline is described in figure 5.5. Twelve hours of videos were recorded per camera for a total of 59 hours (one hour had to be removed because of roadworks), which amounts to about 10000 datapoints available for training and testing. Each datapoint covers 21 seconds of recording, with an overlap (about 1 second) with previous and subsequent datapoints. For each of the datapoint, we collect the q and o values (although in this work we only use the q values so as to restrain data variability and simplify the adaptation task), as well as the RGB, MV and residual frames. Note that the residuals were not directly extracted from the video flux and are in the RGB space, not in their frequency representation.

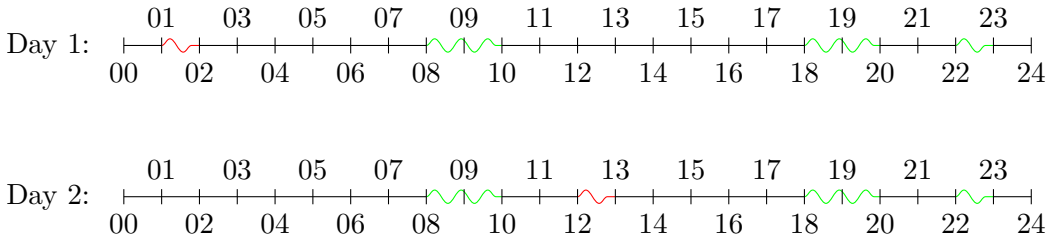
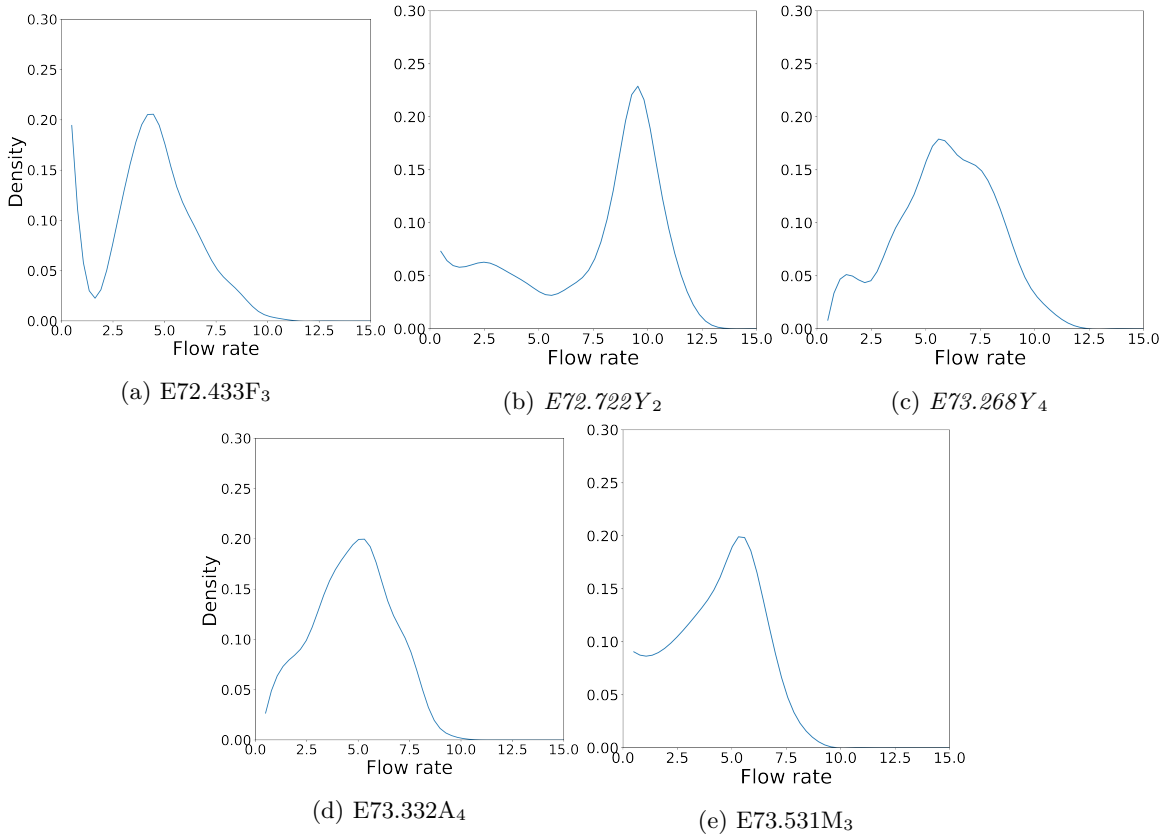


Figure 5.5 – Timelines of the recorded videos. Recording time periods common to the two days are in green and in red are the recordings that were done on only one day.

Camera ID	Orientation		Number of lanes	Is outside
	Coming In	Driving Away		
E72.433F$_3$		X	3	
E72.722Y$_2$	X		2	Yes
E73.268Y$_4$	X		4	
E73.332A$_4$		X	4	
E73.531M$_3$		X	3	

Table 5.1 – Details for each of the available cameras.

Out of the five recorded cameras, three look at the back of the cars and two look at them approaching. The number of lanes are not identical between the cameras, with one camera with two lanes, two with three lanes and two with four lanes. Finally, one camera (*E72.722Y*$_2$) is at the entry of a tunnel and is therefore subject to night and day illuminations.



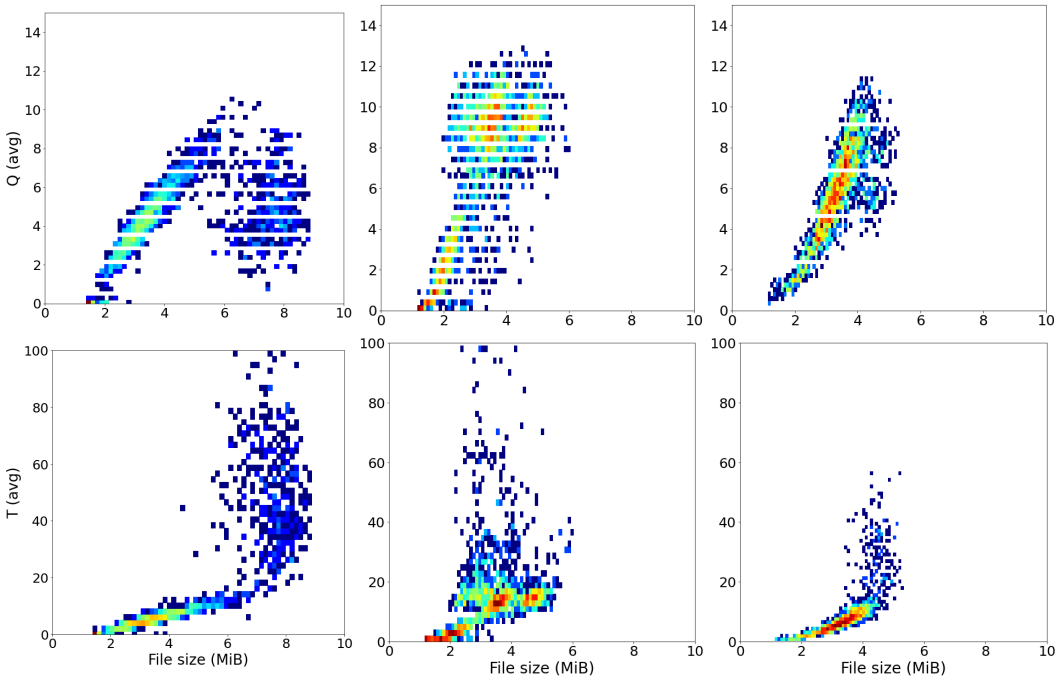
(a) E72.433F$_3$ (b) *E72.722Y*$_2$ (c) *E73.268Y*$_4$

(d) E73.332A$_4$ (e) E73.531M$_3$

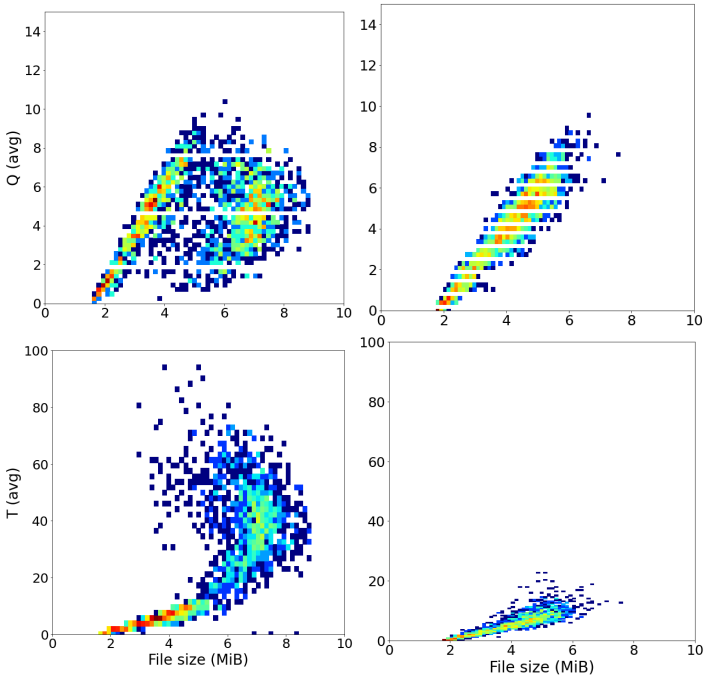
Figure 5.6 – Distributions of the q values for each of the cameras.

The specificities of the cameras are detailed in Table 5.1. Note that, a camera tag is in italic if it looks at car approaching and that the number in index indicates the number of lanes. Regarding the distributions of the q values, we can see in Figure 5.6 that they vary between the cameras. In particular, the mode varies between approximately 5 and 10. Moreover, for some of the cameras, there is a second spike near 0. It is likely that, such differences between the distributions, can cause generalization problem of investigated models.

Finally, we study q and o distributions, depending on the size of the 21 seconds long RGB video files of each sample. The plots for each of the five cameras are provided in Figure 5.7. All the cameras follow a somewhat similar pattern. In particular, for cameras E72.433F$_3$ and E73.332A$_4$ (first and fourth), the first part of the plot is linear in regards to the file size and then becomes noisy. Such plot is very similar to the fundamental diagram of traffic flow theory from Figure 5.3 (bottom left). In the first half of the plot, the traffic is in free flow and both the flow rate and the occupancy grow linearly. In free flow, adding a car in the traffic flow has no impact on the other cars and, as such, the file size increases by a given amount for each added car. Then, when the vehicle flow reaches ρ_c the road density at maximum capacity, it transitions into a congested state, where the flow rate start decreasing and the occupancy explodes. As each car stays longer and longer in vision of the camera, the file size keeps increasing while the reduced vehicles speed induces the reduction in flow rate and augmentation in occupancy. For cameras *E73.268Y*$_4$ and E73.531M$_3$ (third and fifth plots) the data recorded only cover traffic in free flow. Finally, for camera *E72.722Y*$_2$ we see that the flow rate does not follow the same linear relation with the file size. In particular, values of flow rate of 10 are associated with a file size roughly ranging from 3 to 5 MiB. This can be explained by the fact that the camera is located outside. Where tunnel's camera have constant light due to artificial illumination, the outside camera is subjected to night and day illuminations (c.f Figure 5.8), likely inducing changes in the input distribution. This fact raises some questions about transferability from any trained method (based on inside tunnel



(a) E72.433F$_3$ (b) *E72.722Y$_2$* (c) *E73.268Y$_4$*



(d) E73.332A$_4$ (e) E73.531M$_3$

Figure 5.7 – Heat map plot of the q and o values for each of the cameras against the 21 seconds RGB video files' sizes. Color scale is logarithmic, blue means little datapoints and red means a high concentration of datapoints. MiB is Mebibyte.

cameras) to outside cameras.

(a) Reflects due to the sun position and dirtying of the lens.

(b) Typical day configuration

(c) Typical night configuration

(d) Intense illumination probably due to the car having headlights full on

Figure 5.8 – Examples of illumination changes in camera $E72.722Y_2$ caused by its outside positioning. We can see multiple sources of noise, from reflections (a) to intense illumination due to the late night hour (d).

5.2 Flow rate estimation from compressed MPEG4 part-2 videos: Application to Actemium's tunnel dataset

We now seek to apply the flow rate estimation methods proposed in chapter 4 to Actemium dataset. As a reminder, our end goal is to elaborate a flow rate estimation method, leveraging the tunnel cameras in order to provide with an alternative for the costly induction loops. In chapter 4 we have defined our problem as the estimation of a function h which takes sequences of frames $\mathbf{x} \in \mathcal{X}$ as input and that outputs the target value $y \in \mathcal{Y}$:

$$y = h(\mathbf{x}) + \epsilon, \tag{5.6}$$

where ϵ is the measurement error. Based on this formulation, we have proposed regression networks (*DeepMotionCLF*, *DeepMotionCLS*, *DeepMotion3D*, see subsection 4.2.2) using sequences of MV frames as input. The usage of MV frames allows to greatly reduce the size of networks and therefore to greatly speed up the prediction step. Recasting the regression problem as a temporal classification problem, we have proposed CTC-based networks (*DeepMotionCLF-CTC*, *DeepMotion3D-CTC*, see subsection 4.2.3) that rely on a segmenting and counting principle using the CTC loss. All the proposed models were tested on a synthetic dataset. However, these methods have neither been tested on real data, nor compared to a more classical RGB method based on a detection-tracking-estimation pipeline.

Application to real data raises multiple questions. For instance, although the synthetic dataset allows to test variations of the inputs (camera orientation, objects' scale, number of flows), it does not account for the noise inherent to the real conditions (illumination, dirtying, occlusions). Whether the proposed networks can learn to estimate flow rate, given such noise affecting a camera, is an open question. Moreover, cameras are subjected to both setting modifications (number of lanes, orientation, scale) and varying types of noise (illumination, dirtying of the len, camera movements due to trucks, etc.). For instance, cameras facing vehicles and cameras looking at the rear of vehicles will not suffer the same type of illuminations (see Figure 5.8). Therefore, this raises the question of whether adding data from various cameras during training, without accounting for such variations, will help or impede the learning of the networks. Finally, generalization capabilities of the models are to be tested, as a large number of cameras are left without annotations. If we want to leverage

each camera, the proposed networks need to be resilient to unseen settings. To summarize, through the experiments, we aim to answer the following questions:

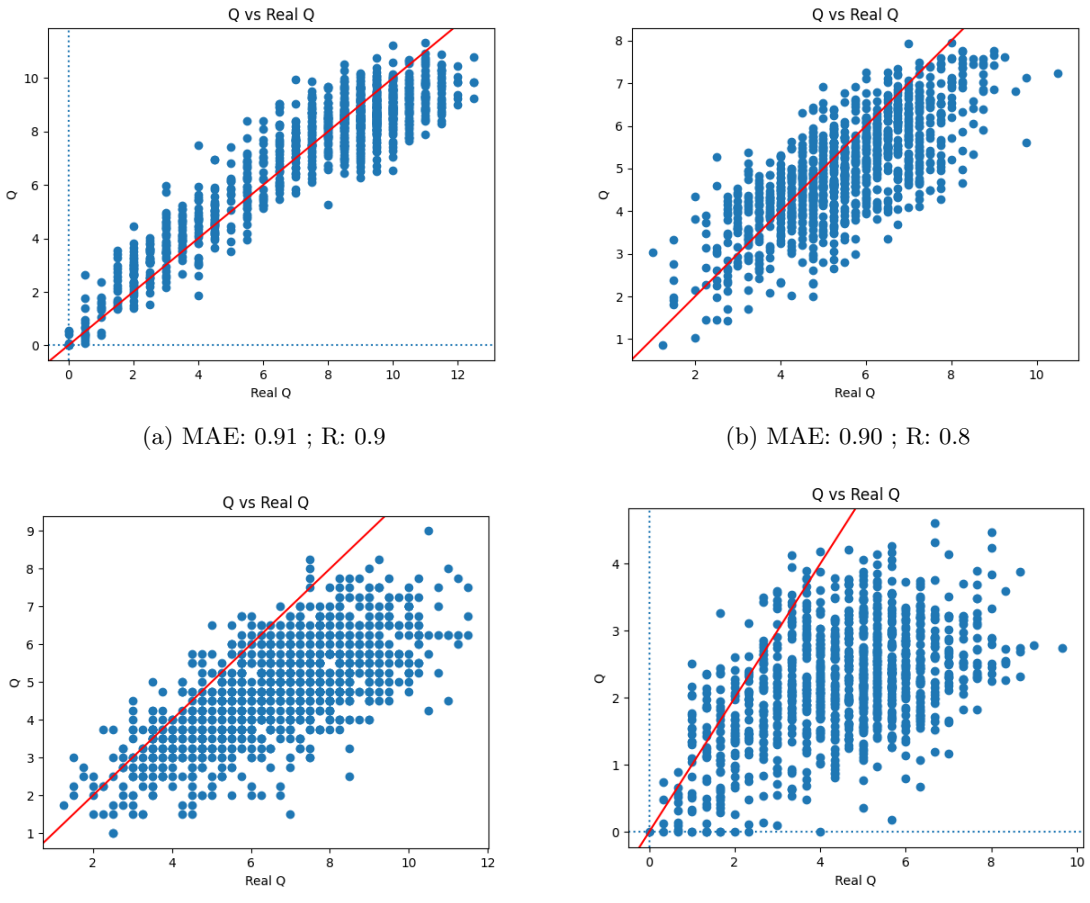
1. Do the methods proposed in chapter 4 allow to estimate the flow rate values when trained on a single camera?

2. How do the MV-frames-based models compare with a RGB detection-tracking-estimation method?

3. Does training on multiple cameras with varying sources of noises and configurations improve or impede the overall accuracy?

4. Do the MV-frames-based models generalize well to unseen cameras?

Depending on the reached accuracy, the proposed methods can have different purposes. For instance, a prediction accurate to the vehicle might be used to raise alert in a critical surveillance system. Conversely, an estimation network solely capable to follow the trend of the traffic flow rate may be used for a more global analysis of the traffic flow. In the presented work we aim to address the second, less constrained, application. To assess the quality of the models, we use the Mean Average Error (MAE) as a performance measure and set the acceptable maximum error to 1. Moreover, to account for the capacity of the model to follow the traffic flow trend over time, we aim for a correlation coefficient between predicted values and real values of at least 0.8. For a better understanding of the selected values, Figure 5.9 shows the plots of predicted vs target values at different MAEs and correlation coefficients. We see that only figure (a) allows for an accurate trend following.



(a) MAE: 0.91 ; R: 0.9

(b) MAE: 0.90 ; R: 0.8

(c) MAE: 1.93 ; R: 0.7

(d) MAE: 2.34 ; R: 0.5

Figure 5.9 – Visual representation of multiple values of MAE and correlation coefficients R.

The rest of the section is divided as follows. First, in order to produce with a RGB-based baseline, we introduce a detection-tracking-estimation paradigm and detail the prediction results. Then, we present the experimental results and their comments for the methods proposed in chapter 4. Finally, as we show limitations on the generalization capacities of the proposed networks, we study the use of domain adaptation so as to palliate the issue of cameras that cannot be annotated.

5.2.1 Baseline: Detect and Track

We start by introducing a baseline RGB method that will be used as reference.

Description of the method The proposed model is based on the classical detection-tracking-estimation pipeline. As the sought model is intended to operate in a real-time setting (possibly on a high number of cameras), the detector needs to be fast and to have low memory consumption. To that extent, we select the well known SSD detector [Liu et al., 2016]. Regarding the tracking system, various solutions exist, the most renown being Sort [Bewley et al., 2016] (a combination of a Kalman Filter [Kalman, 1960] for object displacement estimation and the hungarian algorithm[3] [Kuhn and Yaw, 1955] for object association between frames) and its recent deep update DeepSORT [Wojke et al., 2017]. Because we are looking for a method easy to deploy on cameras that may have different setups (angle, distance to road, etc.), we choose to solely relies on the hungarian algorithm, with euclidean distance, so as to minimize the required manual calibrations. Finally, for the estimation step we rely on a simple emulation of induction loops to avoid cumbersome camera calibrations: for each camera, we manually set a virtual induction loop and count the number of detected tracks crossing the loop to estimate q. The overall procedure is detailed in Figure 5.10 and the detailed algorithms used for the detection, tracking and estimation are provided in Appendix D.

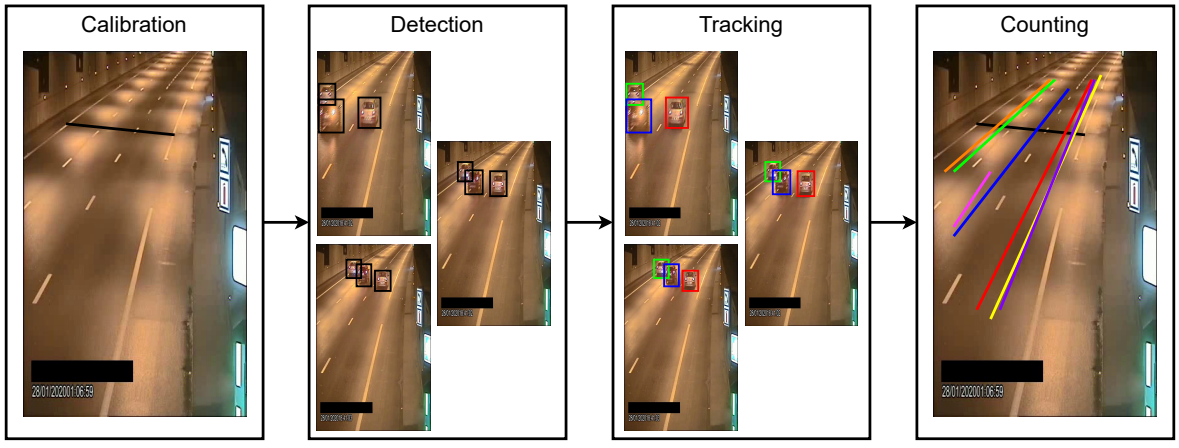


Figure 5.10 – Illustration of the RGB processing pipeline. First, the camera is manually calibrated by drawing an "induction loop" (black line on the left image). Then, each frame is processed by the detector, after which the detections are associated into tracks using the hungarian algorithm. Finally, the tracks are used to compute the flow rate.

Implementation details Regarding the detection networks, we re-use the (RGB) SSD network trained on Actemium detection dataset (c.f chapter 3). The threshold for the selection of the detected boxes is set to 0.5. The emulated induction loops are manually set

[3]The hungarian algorithm is a method to find the best coupling of objects, while minimizing the cost of associating two objects (for instance associating detections at subsequent frames while minimizing the distance between each detection's bounding box).

for each camera. Although no training is performed using the traffic flow dataset, we only evaluate the method on the second day of recording for fair comparison with methods based on MV frames. We detail the results camera per camera. We use the Mean Absolute Error (MAE) and correlation coefficient with the discussed limitations (Figure 5.10) to evaluate the method. For clarity of the notations, correlation coefficients are provided as exponent (in italic) to the MAE results (i.e $0.91^{0.9}$).

As the real time ability is needed, the number of FPS and Datapoints Per Second (DPS) that can be processed are also evaluated. A NVIDIA GTX 1080 is used for computation. The DPS and FPS are computed as the average inference time over 20 datapoints.

Achieved performances The obtained results are summarized in table 5.2. Apart from camera *E73.268*Y_4, we see that for all the cameras we reach the desired performances. Even camera *E72.722*Y_2, which is outside, has a MAE lower than one. Regarding camera *E73.268*Y_4, two main reasons can justify the reported accuracy. First vehicles are seen from the front, leading to camera illumination by the headlights and second, the monitored road has 4 lanes subjected to occlusions due to the positioning of the camera (see Figure 5.6c). Finally, regarding speed, we can see that the method can process up to approximately 3 video flux in parallel at 69 FPS (videos are recorded at 25 FPS).

	E72.433F_3	*E72.722*Y_2	*E73.268*Y_4	E73.332A_4	E73.531M_3	DPS	FPS
SSD-hungarian	$0.66^{0.9}$	$0.90^{0.9}$	$1.27^{0.9}$	$0.58^{0.9}$	$0.61^{0.9}$	0.13	69

Table 5.2 – Mean Average Error for the RGB-based estimation method. Correlations between real and predicted values are provided as exponent to the MAE values. Higher errors are highlighted using a more vivid orange. DPS is Datapoints Per Second (525 frames representing 21 seconds of video) and FPS Frames Per Second.

5.2.2 Estimation from the compressed MPEG4 part-2 representation

We now detail the empirical results on the compressed representation of the tunnel video recordings. To answer the four raised questions, we evaluate the models proposed in chapter 4 by training them on each camera separately. These results are used to evaluate the feasibility of the method on real data as well and to set a comparison basis with the RGB-based model (first and second question). Networks are trained on multiple cameras to show the impact of cameras with multiple settings on the overall accuracy (third question). Speed and memory consumption of the various architectures are finally compared. Generalization capabilities of the networks (fourth question) are evaluated throughout all experiments.

Implementation details Three of the five considered networks in chapter 4 (*DeepMotionCLS*, *DeepMotion3D* and *DeepMotion3D-CTC*) are trained on the Actemium dataset. However, the architectures are modified to fit the corresponding input shape ($525 \times 36 \times 22 \times 2$ instead of $500 \times 13 \times 13 \times 2$ in chapter 4). These modifications are detailed in Table 5.3. Note that we choose to drop the frame-based architectures (*DeepMotionCLF* and *DeepMotionCLF-CTC*) due to the difficulties to train them, as well as their too high memory usage and inference speed limitations (see chapter 4, section 4.2.5).

The networks are trained on a single GPU. For each of the five cameras, E72.433F_3, *E72.722*Y_2, *E73.268*Y_4, E73.332A_4 and E73.531M_3, we use the first day for training and validation (for a gross total of respectively 3,982 and 995 datapoints, or about 800 and 200 per camera) and the second day for testing (5,233 datapoints in total, or about 1,050 datapoint per camera). We use the Mean Squared Error (MSE) as training loss so as to reduce large prediction errors. No data-augmentation is performed and no weight decay is applied. We use an Adam optimizer [Kingma and Ba, 2015] for each training and set the batch size to 32.

DeepMotionCLS		DeepMotion3D		DeepMotion3D-CTC	
Output Size	Layers	Output Size	Layers	Output Size	Layers
$21 \times 25 \times 36 \times 22 \times 2$	input layer	$525 \times 36 \times 22 \times 2$	input layer	$525 \times 36 \times 22 \times 2$	input layer
$21 \times 12 \times 17 \times 10 \times 64$	$[3 \times 3 \times 3, 64$ (s2)]	$262 \times 17 \times 10 \times 64$	$[3 \times 3 \times 3, 64$ (s2)]	$262 \times 17 \times 10 \times 64$	$[3 \times 3 \times 3, 64$ (s2)]
$21 \times 5 \times 8 \times 4 \times 64$	$[3 \times 3 \times 3, 64$ (s2)]	$130 \times 8 \times 4 \times 64$	$[3 \times 3 \times 3, 64$ (s2)]	$130 \times 8 \times 4 \times 64$	$[3 \times 3 \times 3, 64$ (s2)]
$21 \times 1 \times 8 \times 4 \times 64$	AvgPool-$[5 \times 1 \times 1]$	$64 \times 3 \times 1 \times 64$	$\mathbf{[3 \times 3 \times 3, 64}$ (s**2**)]	$64 \times 3 \times 1 \times 64$	$\mathbf{[3 \times 3 \times 3, 64}$ (s**2**)]
$1 \times 1 \times 6 \times 2 \times 64$	CL-$[\mathbf{3 \times 3}, 64]$	$32 \times 1 \times 1 \times 64$	AvgPool-$[\mathbf{2 \times 3 \times 1}]$	$64 \times 1 \times 1 \times 64$	AvgPool-$[\mathbf{1 \times 3 \times 1}]$
$1 \times 1 \times 1 \times 1 \times 768$	Reshape	$1 \times 1 \times 1 \times 2048$	Reshape	-	-
$1 \times 1 \times 1 \times 1 \times 64$	FC-64	$1 \times 1 \times 1 \times 64$	FC-64	$64 \times 1 \times 1 \times 64$	(FC-64) $\times n_{lanes}$
prediction	FC-1	prediction	FC-1	prediction	(FC-2) $\times n_{lanes}$

Table 5.3 – Updated networks for the new input shape. Changes are highlighted in bold. *DeepMotionCLS* changes the size of the ConvLSTM kernel and the dimension of the ConvLSTM output representation (768 vs 64 for Moving Digit dataset). *DeepMotion3D* and *DeepMotion3D-CTC* have an added Conv3D layer and small changes in the pooling layer.

We also evaluate the speed of the networks in both DPS and FPS. We use a NVIDIA GTX 1080, set the batch size to 8 and run 1000 predictions. As the proposed architectures are extremely fast, we preload the datapoints into memory to avoid the input bottleneck. The final FPS value is the average over the total number of predictions (the higher, the better).

Single camera learning We train the three networks referred in Table 5.3 on a given camera and evaluate their performances on the videos issued from the same camera. The results are detailed in Table 5.4.

Overall, the average MAE shows that the proposed models perform well on real data. For the regression architectures, *DeepMotionCLS* and *DeepMotion3D*, errors are lower than for the RGB approach and more stable (lower standard deviation). Results on the CTC-based architecture are more mitigated, with an increase in error, lower correlation between real and predicted values, and higher standard deviation.

	E72.433F$_3$	*E72.722Y$_2$*	*E73.268Y$_4$*	E73.332A$_4$	E73.531M$_3$	Average error
detection-tracking-estimation:						
SSD-hungarian	$0.66^{0.9}$	$\mathbf{0.90}^{0.9}$	$1.27^{0.9}$	$\mathbf{0.58}^{0.9}$	$0.61^{0.9}$	$0.80^{0.9} \pm 0.26$
Regression-based:						
DeepMotionCLS	$\mathbf{0.57}^{1.0}$	$0.91^{0.9}$	$\mathbf{0.68}^{0.9}$	$0.64^{0.9}$	$\mathbf{0.44}^{\mathbf{1.0}}$	$\mathbf{0.65}^{0.9} \pm 0.15$
DeepMotion3D	$0.63^{0.9}$	$1.12^{0.9}$	$0.83^{0.9}$	$0.74^{0.8}$	$0.56^{0.9}$	$0.78^{0.9} \pm 0.20$
CTC-based:						
DeepMotion3D-CTC	$0.75^{0.9}$	$1.03^{0.9}$	$1.93^{0.7}$	$1.91^{0.7}$	$2.15^{0.9}$	$1.55^{0.8} \pm 0.56$

Table 5.4 – Mean Absolute Error and correlation coefficient (between real and predicted values) for each of the architectures trained on each camera separately. For clarity of the notations, correlation coefficients are provided as exponent. Higher errors are highlighted using a more vivid orange. The MV frames-based regression architectures provide with results on par with the RGB-based solution. Results for *DeepMotion3D-CTC* are not satisfactory with an average error largely above the objective threshold of 1.

In more details, when compared to the RGB-based method, *DeepMotionCLS* achieves the best performances. The *DeepMotion3D* architecture is not far behind, with overall similar accuracy and correlation values. Regarding the *DeepMotion3D-CTC*, we can see that the MAE grows up to 2.15 and that the correlation coefficient decreases to 0.7. These results show the difficulty of the CTC network to correctly learn to detect and count vehicles. Such difficulty in training may be due to the imprecisions within the annotations. As we had to match each recorded flow rate value (computed over 20 seconds) with 21 seconds long videos during the annotation process, it is possible that more vehicles are visible than the ones accounted for in the annotations. Because the CTC learns to identify each vehicle, discrepancies between the number of vehicles on screen and the one annotated may lead to

an ill optimization. The network must simultaneously learn to detect the genuine vehicles while ignoring the added ones.

	Source	Target			
	E72.433F$_3$	*E72.722Y*$_2$	*E73.268Y*$_4$	E73.332A$_4$	E73.531M$_3$
Regression-based:					
DeepMotionCLS	**0.57**$^{1.0}$	3.91$^{0.3}$	2.83$^{0.1}$	2.15$^{-0.1}$	2.34$^{0.5}$
DeepMotion3D	0.63$^{0.9}$	**3.47**$^{0.7}$	**2.09**$^{0.3}$	**1.33**$^{0.6}$	**1.03**$^{0.7}$
CTC-based:					
DeepMotion3D-CTC	0.75$^{0.9}$	-	-	-	2.15$^{0.4}$

Table 5.5 – MAE and correlation between the real and predicted flow rate for the networks trained on camera E72.433F$_3$ acting as source data. The correlation coefficients are provided as exponent to the MAE. Higher errors are highlighted using a more vivid orange. Results on other target cameras are obtained without fine-tuning. Due to the limitations of the CTC-based architecture, only the camera with a similar number of lanes as the source is considered.

We now seek to partially answer our fourth question and detail the capacity of the network to generalize to unseen data. We select the networks trained on each camera and report the accuracy performances on the other cameras. In this section we detail the results for camera E72.433F$_3$ (Table 5.5) as it is a camera with an intermediate number of lanes. Results for the training on the other cameras are available in the appendices, (Table D.1). Note that the *DeepMotion3D-CTC* architecture can only be evaluated on cameras with similar number of lanes, i.e E73.531M$_3$. Interestingly, while *DeepMotionCLS* shows the best results for single camera training (Table 5.4), *DeepMotion3D* has the best generalization capability. Still, the results, regardless of the target camera orientation (similar [E73.332A$_4$ and E73.531M$_3$] or inverse [*E72.722Y*$_2$ and *E73.268Y*$_4$] to the source one), do not meet the targeted accuracies (i.e MAE under 1 and correlation above 0.8). In particular, for *DeepMotionCLS* on camera E73.332A$_4$ the correlation between the real and predicted values drops to -0.1. Finally, note that the CTC-based network provides the same level of generalization than the other architectures.

	Source	Target				
	Average	E72.433F$_3$	*E72.722Y*$_2$	*E73.268Y*$_4$	E73.332A$_4$	E73.531M$_3$
Regression-based:						
DeepMotionCLS	**0.69**$^{0.9}$ ± 0.15	1.70$^{0.9}$	3.50$^{0.7}$	2.94$^{0.6}$	**1.00**$^{0.6}$	1.72$^{0.8}$
DeepMotion3D	0.83$^{0.9}$ ± 0.29	**1.56**$^{0.9}$	**3.23**$^{0.7}$	**2.93**$^{0.7}$	1.69$^{0.6}$	**1.57**$^{0.8}$
CTC-based:						
DeepMotion3D-CTC	-	-	-	-	-	-

Table 5.6 – Mean Absolute Errors and correlation coefficients (between real and predicted values) when training on all the cameras but one and predicting the flow rate on the camera left out. The correlation coefficients are provided as exponents to the MAE. Higher errors are highlighted using a more vivid orange. The first column is the average error on the source data. The remaining columns are the errors for the left out camera.

Multiple camera learning Experiments in the setting where multiple cameras are used at training time are now considered. Training are first carried in a leave-one-camera-out setting. Results are compiled in Table 5.6. First, we can notice a slight degradation of the results on the source data (first column). However, this slight increase in MAE is to be mitigated as the results are still within the targeted performance bounds. These findings are important as they show that data from several cameras can be gathered to form the training set, without impeding prediction performances (third question). Regarding the target cameras, we can see large improvements for the correlation coefficients, with the minimum value going from

0.1 (Table 5.5) to 0.6 (Table 5.6). However, such results are still not high enough to provide with reliable predictions on unseen cameras. Finally, we note a tendency for the cameras looking at the rear of the vehicles (E72.433F$_3$, E73.332A$_4$ and E73.531M$_3$) to provide with a lower MAE. Such results could be explained by the absence of the headlights, hence reducing illumination issues. Overall, on the source data, the results are in acceptable range, however on the target data, none of the results are satisfactory.

	Source	Target		
	Average	E72.433F$_3$	E73.332A$_4$	E73.531M$_3$
Regression-based:				
DeepMotionCLS	$\mathbf{0.59}^{0.9} \pm 0.09$	$2.95^{0.8}$	$\mathbf{1.00}^{0.7}$	$\mathbf{1.48}^{0.8}$
DeepMotion3D	$0.67^{0.9} \pm 0.08$	$\mathbf{2.74}^{0.8}$	$1.97^{0.8}$	$1.60^{0.7}$
CTC-based:				
DeepMotion3D-CTC	$0.63^{0.9} \pm 0.12$	$3.67^{0.4}$	-	$2.15^{0.4}$

Table 5.7 – MAE and correlation between the real and predicted flow rates when training on all the cameras looking at the rear of the vehicles (E72.433F$_3$, E73.332A$_4$, E73.531M$_3$) but one and predicting on the left-out. Higher errors are highlighted using a more vivid orange. The first column is the average MAE error for each of the source combinations, the remaining ones are the errors for the left out camera. The correlation coefficients are provided as exponents to the MAE. For *DeepMotion3D-CTC* only cameras with three lanes were used for adaptation due to the limitations of the architecture.

	Source	Target	
	Average	*E72.722Y$_2$*	*E73.268Y$_4$*
Regression-based:			
DeepMotionCLS	$\mathbf{0.90}^{0.9} \pm 0.11$	$4.78^{0.8}$	$2.90^{0.4}$
DeepMotion3D	$0.98^{0.9} \pm 0.15$	$\mathbf{4.40}^{0.6}$	$\mathbf{1.39}^{0.6}$
CTC-based:			
DeepMotion3D-CTC	-	-	-

Table 5.8 – MAE and correlation between the real and predicted flow rates when training on one of the two cameras looking at the front of the vehicles (*E72.722Y$_2$*, *E73.268Y$_4$*) and predicting on the remaining one. Higher errors are highlighted using a more vivid orange. The first column is the average error for each of the fold, the remaining ones are the errors for the left out camera. The correlation coefficients are in exponent of the MAE.

Given the differences in accuracy between the groups of cameras with different orientations, we now separate the dataset by camera orientation. The hope being that this will avoid to train the networks on too dissimilar data and help them learn a better representation. As for the previous experiments training are carried on all the cameras of similar orientation but one and evaluated on the one left-out. Table 5.7 and Table 5.8 include the empirical evaluations for respectively rear and front view of the vehicles. Comparing Table 5.7 and Table 5.6, we see that removing cameras *E72.722Y$_2$* and *E73.268Y$_4$* from the training set helps improving the correlation coefficients for the regression-based networks (from 0.6 minimum to 0.7 minimum). However, accuracy is not preserved. For instance on camera E72.433F$_3$, for the architectures *DeepMotionCLS* and *DeepMotion3D* the error goes from respectively 1.70 and 1.56 to 2.95 and 2.74. Similarly, for the cameras looking at the front of the vehicles (Table 5.8), results either show a decrease in the correlation coefficient or an increase in MAE. Finally, the CTC architecture, *DeepMotion3D-CTC*, presents with the worst results, that is a correlation coefficient of 0.4 and a MAE superior to the ones of other architectures. However, this result is to be mitigated as it is obtained using only one camera as source due to the limitations imposed by the CTC architecture. Overall, removing cameras with changing properties does not seem to help with the generalization issue.

On the prediction speed We now study the speed of the newly created architectures and compare them with the RGB baseline. Results are provided in Table 5.9 and the accuracies are the average of test errors while training on each camera separately. Regarding the accuracies, we can see that the regression-based approaches (*DeepMotionCLS* and *DeepMotion3D*) provide with more accurate and less varying results. Moreover, regarding prediction speed, compressed-data-based architectures are far better, as they are at least 2000 times faster (*DeepMotionCLS*) than the RGB network and at most 3200 times faster (*DeepMotion3D*). Furthermore, compressed representation networks require far less memory. Finally, we note that the architecture *DeepMotion3D-CTC*, while providing with impressive speed has the highest MAE at 1.55 and is the less stable with a standard deviation of 0.56.

Network	MAE	GPU Memory (MiB)	DPS	FPS
SSD300	$0.80^{0.9} \pm 0.26$	7,805	0.13	69
Regression based:				
DeepMotionCLS	$\mathbf{0.65}^{0.9} \pm 0.15$	703	271	142,472
DeepMotion3D	$0.78^{0.9} \pm 0.20$	703	505	265,095
CTC based:				
DeepMotion3D-CTC	$1.55^{0.8} \pm 0.56$	703	418	219,308

Table 5.9 – Speed comparison of the various architectures. The correlation between the predicted and the real values is noted as exponent to the MAE. Higher errors are highlighted using a more vivid orange. Batch size is set to 8 for the evaluations. All of the networks are much faster than any of the detector seen so far.

Synthesis We have identified four questions to be answered, of which if the methods proposed in chapter 4 allow to estimate the flow rate values when trained on a single camera and how the MV-frames-based methods compare with an RGB detection-tracking-estimation method. In this first set of experiments, we have shown that the proposed architectures using the compressed representation can be used for flow rate estimation. In particular, for the regression-based architectures, accuracy is better than the RGB-based method while providing with impressive speed gains, up to $\times 3200$. However, the CTC based architecture shows unsatisfactory results, not meeting the targeted accuracy with a MAE of 1.55.

Regarding whether training on multiple cameras with varying sources of noise and configurations would improve or impede the overall accuracy, Table 5.6 does not show improvements of the accuracy. However, as both the MAE and the correlation coefficients fulfill the target bounds, it is safe to assume that data from different cameras can be gathered in the training set without degrading performances.

The last concern is how the MV-frames-based methods generalize to unseen cameras. Tables 5.5, 5.6, 5.7 and 5.8 show that the trained networks do not generalize in a consistent manner to unseen data from other cameras. Although the final accuracies are not satisfactory enough yet, training with data from multiple cameras seems to help with the predictions (see Table 5.6). Such results potentially may lead to significant performance improvements if a larger, more diverse dataset was to be collected.

5.2.3 Domain Adaptation towards unseen cameras

Previous section shows estimation of the flow rate from traffic recording cameras can be efficiently achieved using the proposed MV based networks. These models being extremely light and fast, real time processing is achievable, which is of great interest for industrial purpose. However these architectures have one main limitation: they do not generalize well to unseen cameras. Given the difficulty to acquire labelled training data, we study the possibility to apply a domain adaptation framework to deal with shifts in joint input and output distributions across cameras. For that, we exploit the DeepJDOT method [Damodaran et al.,

2018] (see chapter 4, subsection 4.3.1 for the details). We first run the adaptation from one camera to the others so as to see if any pattern emerges. Then, we run adaptation on the networks trained on multiple cameras. The latter experiment aims to test the resilience of the adaptation to training data from multiple sources, as well as to find out if added variation in the source dataset allows for improved results. As DeepJDOT aims to reduce the joint input (embedding) and output distribution of the source and target datasets based on optimal transport, it works best with large batch size. However, due to memory constraints, we restrict the batch size to 32. We also set the value of $\alpha = 0.01$ (in OT ground distance, Equation 4.3) as we empirically find the value to work best.

	Source		Target							
	E72.433F$_3$		*E72.722Y$_2$*		*E73.268Y$_4$*		E73.332A$_4$		E73.531M$_3$	
Regression-based:										
DeepMotionCLS	**0.57**$^{1.0}$	**0.59**$^{1.0}$	3.91$^{0.3}$	3.98$^{0.5}$	2.83$^{0.1}$	2.43$^{0.6}$	2.15$^{-0.1}$	1.34$^{0.4}$	2.34$^{0.5}$	1.15$^{0.7}$
DeepMotion3D	0.63$^{0.9}$	0.66$^{0.9}$	**3.47**$^{0.7}$	**3.51**$^{0.7}$	**2.09**$^{0.3}$	**2.40**$^{0.7}$	**1.33**$^{0.6}$	**0.94**$^{0.7}$	**1.03**$^{0.7}$	**1.00**$^{0.8}$
CTC-based:										
DeepMotion3D-CTC	0.75$^{0.9}$	0.75$^{0.9}$	-	-	-	-	-	-	2.15$^{0.4}$	1.49$^{0.5}$

Table 5.10 – MAE before (left grayed columns) and after adaptation (right column) when trained on camera E72.433F$_3$ and adapted towards the other cameras (one by one). The correlation between predicted and real flow rates is provided as an exponent to the MAE value. A green cell means improvements after adaptation and a red one means a decrease in accuracy.

Single camera adaptation The networks that were trained on camera E72.433F$_3$ are first adapted towards the other cameras. We select this peculiar camera as it has an intermediate number of lanes and will showcase the behavior on cameras with similar (E73.332A$_4$ and E73.531M$_3$) and opposite orientations (*E72.722Y$_2$* and *E73.268Y$_4$*). Results given in Table 5.10 reveal that adapting towards cameras with opposite orientation worsens the accuracy (3 red cells out of 4). However, this lowering in accuracy is to be mitigated as the correlation coefficients largely improve in most cases. Regarding the two cameras with similar orientation, the accuracy always improves, both in MAE and correlation. In particular, for the architecture *DeepMotion3D*, the results are very close to the targeted accuracy (MAE < 1 and correlation > 0.8). Finally, we can notice, on the source camera, a slight degradation of the accuracy after adaptation. However, these performances on source domain are still in the targeted range. Overall, it seems that cameras are more easily adapted towards cameras with similar orientation.

	Source		Target					
	Average		E72.433F$_3$		E73.332A$_4$		E73.531M$_3$	
Regression-based:								
DeepMotionCLS	**0.59**$^{0.9}$	**0.62**$^{0.9}$	2.95$^{0.8}$	**1.18**$^{0.9}$	**1.00**$^{0.7}$	1.08$^{0.6}$	**1.48**$^{0.8}$	**0.80**$^{0.9}$
DeepMotion3D	0.67$^{0.9}$	0.65$^{0.9}$	**2.74**$^{0.8}$	1.30$^{0.8}$	1.97$^{0.8}$	**0.81**$^{0.8}$	1.60$^{0.7}$	0.94$^{0.8}$
CTC-based:								
DeepMotion3D-CTC	0.63$^{0.9}$	0.63$^{0.9}$	3.67$^{0.4}$	3.46$^{0.5}$	-	-	2.15$^{0.4}$	1.49$^{0.5}$

Table 5.11 – MAE and correlation between the real and predicted values on cameras looking at the rear of vehicles. The networks were trained on two of the three available cameras and tested on the one left out. Grayed columns contain the results before domain adaptation and lowest error on a given camera are in bold. A red cell means a worsening of the error and a green one an improvement. Correlation coefficients are exponent to the MAE values. Apart from one case, adaptation always improve the accuracy of the networks.

Multiple camera adaptation Adaptation from one camera to another shows improvement when the adaptation is performed on cameras with similar positioning (i.e looking at

the rear of the vehicles). However, results are not satisfactory enough yet. We now explore if adding cameras with similar orientations to the training set can help improving the accuracy of the adapted networks.

Let us detail the results for the cameras recording the goings of the vehicles (rear view, Table 5.11). Experiments seem to confirm the intuition that adding similar cameras to the training set helps both training and adaptation. Apart for *DeepMotionCLS* on camera E73.332A$_4$, which slightly decreases the accuracy after adaptation, all the other results of regression-based networks show substantial improvements. Most of the accuracies after adaptation match, at least partially, the targeted precision. *DeepMotionCLS* has two correlation coefficients at 0.9 and *DeepMotion3D* yields two MAE below 1. By comparison, results in Table 5.10 have a maximum correlation coefficient of 0.8 and only one MAE below 1. Regarding the results on the source cameras, they can be considered stable given that the changes before and after adaptation are minimal. Notice that conclusions on the CTC architecture are hard to devise as the adaptation was done from one camera to another due to the lane number restriction of the method. There are improvements, but far from meeting the targeted requirements.

Let us now consider adaptation for the cameras recording the comings of the vehicles (front view, Table 5.12). Again, selecting cameras with similar orientation seems to help during the adaptation process, with improvement in MAE on 3 out of the 4 target ones (only 1 in Table 5.10). Moreover, when the target MAE is not improving (in red), the correlation goes from 0.6 to 0.8, showing that the adapted network better capture the behavior of the traffic flow.

	Source		Target			
	Average		*E72.722Y$_2$*		*E73.268Y$_4$*	
Regression-based:						
DeepMotionCLS	**0.80**$^{0.9}$	**0.83**$^{0.9}$	4.78$^{0.8}$	**2.98**$^{0.8}$	2.90$^{0.4}$	2.30$^{0.7}$
DeepMotion3D	0.98$^{0.9}$	1.01$^{0.9}$	**4.40**$^{0.6}$	3.06$^{0.7}$	**1.39**$^{0.6}$	**2.18**$^{0.8}$
CTC-based:						
DeepMotion3D-CTC	-	-	-	-	-	-

Table 5.12 – MAE before (left grayed columns) and after adaptation (right column) on the cameras viewing the front of the vehicles. Lowest errors on a given camera are in bold. A green cell refers to improvement, otherwise red. Correlation coefficients are exponent to the MAE values.

Overall, results from Tables 5.11 and 5.12 seem to validate the hypothesis that adapting from cameras with only similar orientation helps improving the accuracy. To further test that hypothesis, we now seek to adapt networks trained on all the cameras but one to the left-out camera. As data with different orientations are mixed-up in the source domain, we should not expect improved results as this may penalize the adaptation procedure. Empirical results are given in Table 5.13. For the cameras E72.433F$_3$ and E73.531M$_3$, the adaptation provides with excellent results, with an error below 1 and a correlation fairly high (≥ 0.8). In particular, 3 out of 4 results are the best so far, showing that adding cameras with opposite orientation can help the adaptation. However, for camera E73.332A$_4$ the performances are degraded. Finally, for the cameras viewing at approaching vehicles (*E72.722Y$_2$* and *E73.268Y$_4$*), we also get, out of 4 results, 3 of the best so far (but with fairly high MAE values). Still, such results could be explained by the fact that the two cameras (*E72.722Y$_2$* and *E73.268Y$_4$*) are extremely dissimilar, hence providing with bad adaptation results.

Synthesis In subsection 5.2.2 we highlight difficulty for trained networks to generalize to unseen cameras. To overcome the generalization issue, we have experimented on domain adaptation. The first set of experiments (Tables 5.10, 5.11 and 5.12) show limited benefit of the adaptation but seem to demonstrate that, adaptation towards cameras with similar

	Source Average		Target E72.433F$_3$		*E72.722Y$_2$*		*E73.268Y$_4$*		E73.332A$_4$		E73.531M$_3$	
Regression-based:												
DeepMotionCLS	**0.69**$^{0.9}$	**0.73**$^{0.9}$	1.70$^{0.9}$	**0.70**$^{0.9}$	3.50$^{0.7}$	3.87$^{0.8}$	2.94$^{0.6}$	2.02$^{0.7}$	**1.00**$^{0.6}$	**2.03**$^{0.4}$	1.72$^{0.8}$	0.92$^{0.8}$
DeepMotion3D	0.83$^{0.9}$	0.85$^{0.9}$	**1.56**$^{0.9}$	0.82$^{0.9}$	**3.23**$^{0.7}$	**3.02**$^{0.7}$	**2.93**$^{0.7}$	**1.57**$^{0.8}$	1.69$^{0.6}$	2.08$^{0.5}$	**1.57**$^{0.8}$	**0.88**$^{0.8}$
CTC-based:												
DeepMotion3D-CTC	-	-	-	-	-	-	-	-	-	-	-	-

Table 5.13 – MAE and correlation between the real and predicted values, before adaptation (grayed column, left) and after adaptation (right). The source domain is represented by M-1 cameras and the target domain is the left out camera (M being the total number of cameras). A green cell denotes improvements after adaptation, a red one a decrease in accuracy and a blue one that the obtained result is best out of all the experiments (Tables 5.10, 5.11 and 5.12).

orientation, helps providing with more accurate networks. However, in the last experiment (Table 5.13) the accuracy after adaptation was further improved for some of the cameras (although still not meeting the set accuracies) through the addition of more samples from dissimilar cameras. As adaptation is key to the deployment of the proposed solution, we study in more depth the DeepJDOT adaptation process to understand the root cause of its instability on our data.

5.3 Discussion on Domain Adaptation and DeepJDOT

Domain adaptation experiments show that the effectiveness of the adaptations is not consistent enough across cameras to allow for a reliable industrial usage. Moreover, no clear pattern linking the cameras' settings to the inconsistencies seem to emerge. As in industrial applications the target domain set will not be annotated, there is no proper way to assess correctness of the adaptation. In this section we discuss on these limitations.

5.3.1 The limits of domain adaptation

Let us consider the *DeepMotionCLS* model and the adaptation towards camera E73.531M$_3$ from Table 5.10. The DeepJDOT framework manages to reduce the MAE, from 2.34 to 1.15, and improves the correlation coefficient from 0.5 to 0.7. However, these results are not good enough to avoid large errors. This can be seen in Figure 5.13c, which compares the estimated values with the real flow rate values after DA. The network saturates around 6. This is problematic as it leaves predictions oblivious to a large range of values.

In order to explain such behavior, we remind that DeepJDOT is based on a three part loss (see subsection 4.3.1) that aims to: reduce the discrepancy between the source and target domain embeddings, as well as between the source and target labels, and to maintain the original accuracy on the source set. Therefore, failure in learning to reduce the discrepancies could explain the observed limitations. For that, we visualize the source and target embeddings before (Figure 5.11) and after (Figure 5.12) adaptation using the t-SNE [van der Maaten and Hinton, 2008]. In the plots, the circles are data of the source domain and the crosses correspond to the target domain. The colors indicate the value of q to be predicted, blue being a low value and green a high one. We observe that before adaptation, source and target embeddings are almost apart. Also, the source samples are ordered by level of flow rate contrary to the target embeddings which are more mixed up. After adaptation (Figure 5.12), we see that both the target and source embeddings are now much better matched. However, while the points (source and target) seem ordered from the lowest to the highest, we can notice that the low and high output values from the target domain seem to be mixed up towards mid range source values. This observation is validated by Figure 5.13c, where the network underestimates high values and overestimate the low ones.

To get an insight on such mix up, we consider the output (flow rate) distributions on the source (Figure 5.13a) and target (Figure 5.13b) sets. We see that the source and target

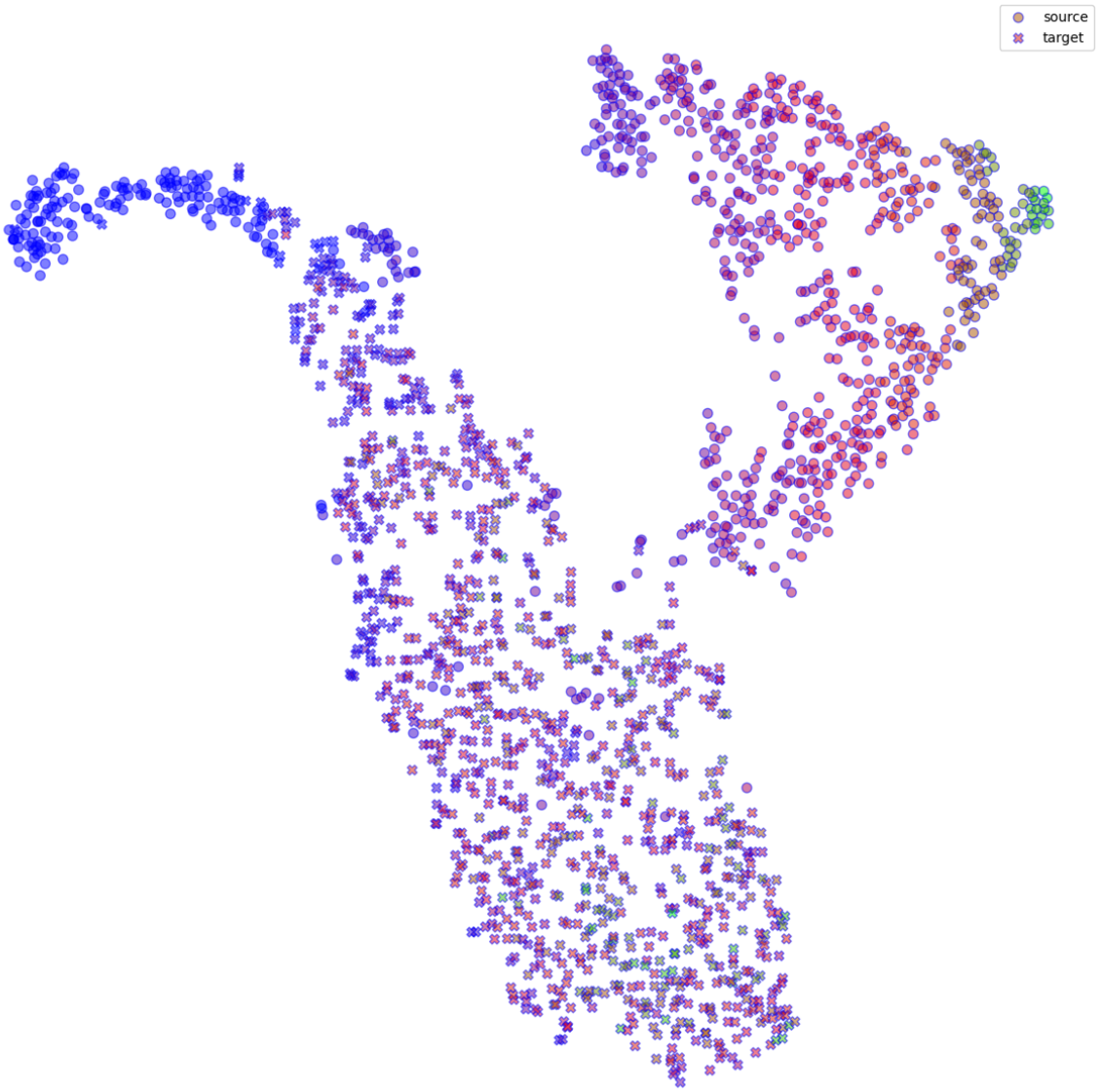


Figure 5.11 – t-SNE plot before domain adaptation. The circles correspond to the source domain and the crosses to the target domain. The colors indicate the value of q to be predicted, blue being a low value and green a high one.

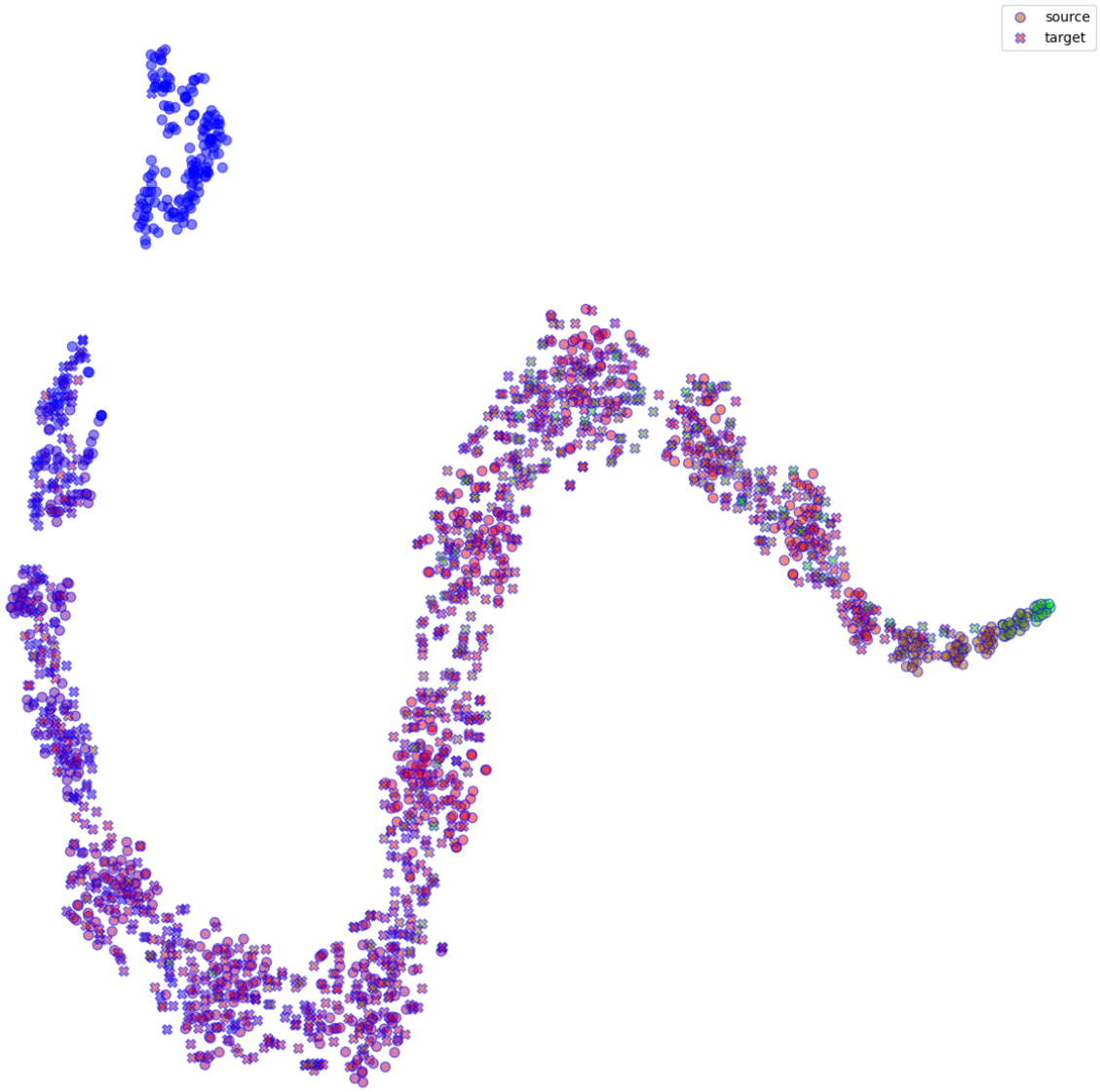
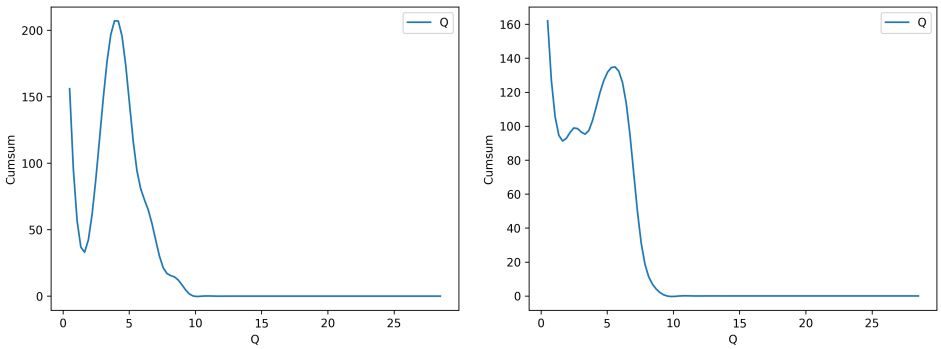
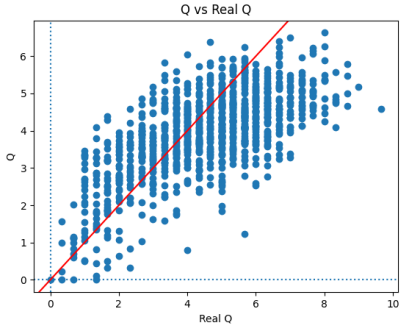


Figure 5.12 – t-SNE plot after domain adaptation. The circles correspond to the source domain and the crosses to the target domain. The colors indicate the value of q to be predicted, blue being a low value and green a high one.



(a) q distribution on the source domain.

(b) q distribution on the target domain.



(c) Scatter plot of the predicted vs. real values on the target domain.

Figure 5.13 – Plots describing the distribution of outputs on the source (a) and target (b) domains as well as the quality of the predictions on the target domain after adaptation (c).

distributions differ greatly. In particular, in the source distribution, the density of values around 1 is very low when compared with the density of the values at 0 and close to 5. In contrast, on the target distribution, the density is much stable across the flow rate values with only a small decrease between 1 and 3. While this may seem anecdotal, DeepJDOT relies on optimal transport to match $(\mathbf{z}^s, y^s)$ and $(\mathbf{z}^t, f(\mathbf{z}^t))$ and then, uses this matching to set the source outputs as proxy for the target embedding outputs. Therefore, if the source and target output distributions strongly differ, target output values will likely be pushed towards lower or higher values. In that case, as the density between 1 and 3 of the target domain is much higher than the one in the source domain, the values in that range get "pushed" towards 5 because of the coupling strategy. Similarly, as the source and target domains do not reach their second density pic at the same flow rate value (respectively before and after 5), high flow rate values of the target domain get "pushed" towards smaller values. This might justify the over and under-estimation in Figure 5.13c.

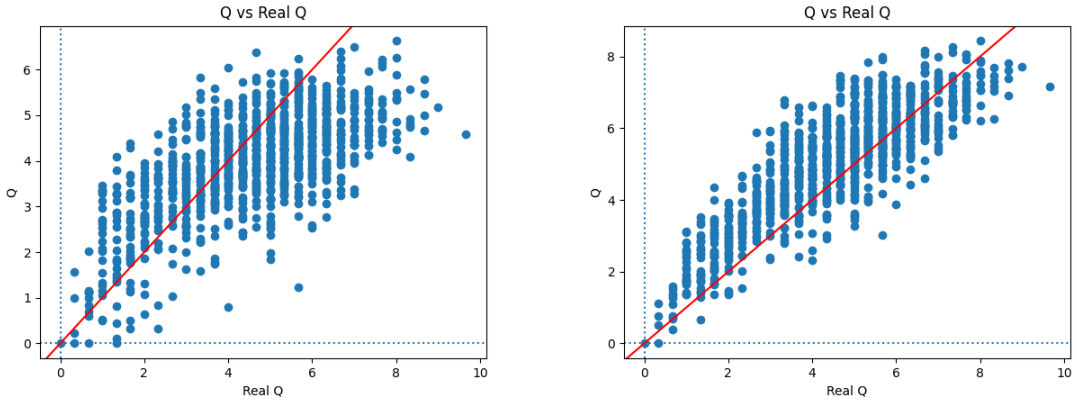
5.3.2 Prediction with oracle

Limitation of domain adaptation applied to our flow rate estimation problem may come from differences in the source and target output distributions. This may impede the matching achieved by optimal transport in the mini-batches. To correct for that, let us assume we have some weak information on the target distribution. Specifically, let us consider we have at disposal the histogram bins of the target domain samples. Using oracle information, we run DA by randomly selecting in the mini-batches the source and target samples, according to the histograms bins, to prevent from output distribution shift. The subsequent experiments are run with the same setups as previously.

Single camera adaptation Similarly to Table 5.11, we run adaptation from camera E72.433F$_3$ to the others. Results are summarized in Table 5.14. We see that the oracle-based adaptation improves the prediction results on *all* the cameras (only 0.01 point drop on the source camera for *DeepMotionCLS*). On the target cameras, the minimal correlation value is now of 0.7 (previously 0.4) and the maximal MAE is of 1.93 (previously 3.98). Regarding the "leaking" effect observed in the discussion, Figure 5.14 shows a comparison side by side of the predicted vs. real values for the adaptation without and with oracle (*DeepMotionCLS* on camera E73.531M$_3$). We can see that the squeezing effect visible in Figure 5.14a tends to disappear in Figure 5.14b. Such results show that discrepancies in the source and target output distribution do impede the adaptation. Still, even with oracle information, for the camera E73.332A$_4$ the correlation value of 0.7 is lower than what we were aiming for. One explanation for such behavior might be a lack of samples in some bins, leading to incorrect estimation. If so, adding more samples should smooth the distributions and provide with improved results. Finally, regarding the CTC network, the adaptation, even with the oracle information is inefficient.

	Source E72.433F$_3$		Target *E72.722Y*$_2$		*E73.268Y*$_4$		E73.332A$_4$		E73.531M$_3$	
Regression-based:										
		($0.59^{1.0}$)		($3.98^{0.5}$)		($2.43^{0.6}$)		($1.34^{0.4}$)		($1.15^{0.7}$)
DeepMotionCLS	$\mathbf{0.57}^{1.0}$	$\mathbf{0.58}^{0.9}$	$3.91^{0.3}$	$\mathbf{1.49}^{0.9}$	$2.83^{0.1}$	$1.40^{0.9}$	$2.15^{-0.1}$	$0.97^{0.7}$	$2.34^{0.5}$	$\mathbf{0.89}^{0.9}$
		($0.66^{0.9}$)		($3.51^{0.7}$)		($2.40^{0.7}$)		($0.94^{0.7}$)		($1.00^{0.8}$)
DeepMotion3D	$0.63^{0.9}$	$0.63^{0.9}$	$\mathbf{3.47}^{0.7}$	$1.93^{0.8}$	$\mathbf{2.09}^{0.3}$	$\mathbf{1.02}^{0.8}$	$\mathbf{1.33}^{0.6}$	$\mathbf{0.91}^{0.7}$	$\mathbf{1.03}^{0.7}$	$0.89^{0.8}$
CTC-based:										
		$0.75^{0.9}$	-	-	-	-	-	-		$1.49^{0.5}$
DeepMotion3D-CTC	$0.75^{0.9}$	$0.75^{0.9}$	-	-	-	-	-	-	$2.15^{0.4}$	$1.20^{0.6}$

Table 5.14 – MAE and correlation results from camera E72.433F$_3$ to the others using the oracle information. Grayed columns contain the results before domain adaptation. For clarity sake, we report the accuracies without oracle information on top of the new ones (in between parenthesis). As before, a green cell means that the results improve on both the source accuracy and the one obtained from the vanilla adaptation, an orange cell means that the results only improves on one of the two values.



(a) Scatter plot of the predicted vs. real values on the target domain without oracle information.

(b) Scatter plot of the predicted vs. real values on the target domain with oracle information.

Figure 5.14 – Plots of the Predicted q vs. real q values, without oracle information (a) and with oracle information (b).

Multiple camera adaptation We now test the proposed oracle adaptation to the networks trained on multiple cameras. Adaptation is first carried out on the networks trained on all the cameras but one. The results are reported in Table 5.15. Similarly, adaptation

with oracle information improves the final accuracy. The only worsening when compared with the previous adaptation is for camera E72.433F$_3$ on *DeepMotion3D*, where the MAE goes up to 1.01 but is still acceptable in industrial environnement. Similar results can be seen on the source cameras, with a slight increase that would have no impact on the final application. Regarding the results overall, the maximum MAE went from 3.87 down to 1.79 and the correlation value from a minimum of 0.4 up to a minimum of 0.7. According to performance measures, 7 errors values out of 10 are below 1 (4 previously) and 7 correlation coefficients are of 0.8 or above (5 previously). Finally, it is important to note that the camera with the worst results is *E72.722Y*$_2$, which is the only one placed outside and with only two lanes. These facts could explain the adaptation difficulties.

	Source		Target									
	Average		E72.433F$_3$		*E72.722Y*$_2$		*E73.268Y*$_4$		E73.332A$_4$		E73.531M$_3$	
Regression-based:												
		(**0.73**$^{0.9}$)		(0.70$^{0.9}$)		(3.87$^{0.8}$)		(2.02$^{0.7}$)		(2.03$^{0.4}$)		(0.92$^{0.8}$)
DeepMotionCLS	**0.69**$^{0.9}$	**0.73**$^{0.9}$	1.70$^{0.9}$	**0.69**$^{0.9}$	3.50$^{0.7}$	**1.66**$^{0.8}$	2.94$^{0.6}$	0.95$^{0.8}$	**1.00**$^{0.6}$	**0.84**$^{0.7}$	1.72$^{0.8}$	0.69$^{0.9}$
		0.85$^{0.9}$		0.82$^{0.9}$		3.02$^{0.7}$		1.57$^{0.8}$		2.08$^{0.5}$		0.88$^{0.8}$
DeepMotion3D	0.83$^{0.9}$	0.85$^{0.9}$	**1.56**$^{0.9}$	1.01$^{0.9}$	**3.23**$^{0.7}$	1.79$^{0.7}$	**2.93**$^{0.7}$	**0.80**$^{0.9}$	1.69$^{0.6}$	0.98$^{0.7}$	**1.57**$^{0.8}$	**0.67**$^{0.9}$
CTC-based:												
DeepMotion3D-CTC	-	-	-	-	-	-	-	-	-	-	-	-

Table 5.15 – MAE and correlation coefficient when the source network was trained on all cameras but one and tested on the remaining one. Adaptation is done using equalized distributions. Grayed columns contain the results before domain adaptation. We report the accuracies without oracle information on top of the new ones (in between parenthesis) and add a color code as before. A green cell means that the results improve on both the source accuracy and the one obtained from the vanilla adaptation, an orange cell means that the results only improve on one of the two values.

We now analyse the results when grouping the cameras by orientation (i.e looking at the rear or front of vehicles). Table 5.16 regroups the results for the cameras looking at the rear of the vehicles and Table 5.17 regroups the results for the cameras looking at the front of the vehicles. Results for the outgoing traffic are first detailed. Again, the accuracy improves for *all* the target cameras (except for the CTC-based architecture which once again behaves poorly). Moreover, the MAE on all the target cameras is lower than the target value of 1. Similarly, the correlation coefficients are all equal or above 0.8. Then, Table 5.17, shows that all the results on the target camera improve. In particular, all the correlation coefficients are equal or above the targeted threshold. However, on the target cameras, 3 adaptations out of 4 produce with MAE values above 1. Moreover, on the source camera, for architecture *DeepMotion3D* we see a worsening of the results. Given that only two cameras are used and that, one of them (*E72.722Y*$_2$) is outside and had shown the worst adaptation accuracy on previous experiments, the results tend to advocate against the addition of outside cameras to training sets.

5.3.3 Synthesis and perspectives

Domain adaptation using the DeepJDOT framework is a complex task that requires to meet a few prior assumptions. We have shown that one requirement of the method is the similarity between the source and target output distributions. In case of discrepancy between these distributions source and target, the samples are wrongly matched, leading to poor accuracy.

Based on oracle information, domain adaptation using DeepJDOT can be largely improved closing the estimation accuracy gap between annotated and unseen cameras. Specifically, using camera recordings with identical orientation seems to perform best. Such results are extremely encouraging as they show that, when adapted with carefully selected data, the proposed regression methods can be used to estimate traffic flow for any tunnel's camera setting.

While obtaining oracle information might seem to be impossible, simple solutions might be accessible. Figure 5.7 suggest that the distribution of the flow rate values roughly follows

	Source Average		Target E72.433F$_3$		Target E73.332A$_4$		Target E73.531M$_3$	
Regression-based:								
DeepMotionCLS	**0.59**$^{0.9}$	(0.62$^{0.9}$) **0.60**$^{0.9}$	2.95$^{0.8}$	(1.18$^{0.9}$) **0.78**$^{0.9}$	**1.00**$^{0.7}$	(1.08$^{0.6}$) 0.83$^{0.8}$	**1.48**$^{0.8}$	(0.80$^{0.9}$) **0.58**$^{0.9}$
DeepMotion3D	0.67$^{0.9}$	(0.65$^{0.9}$) 0.64$^{0.9}$	**2.74**$^{0.8}$	(1.30$^{0.8}$) 0.99$^{0.9}$	1.97$^{0.8}$	(0.81$^{0.8}$) **0.80**$^{0.8}$	1.60$^{0.7}$	(0.94$^{0.8}$) 0.66$^{0.9}$
CTC-based:								
DeepMotion3D-CTC	0.63$^{0.9}$	(0.63$^{0.9}$) 0.63$^{0.9}$	3.67$^{0.4}$	(3.46$^{0.5}$) 3.46$^{0.4}$	-	-	2.15$^{0.4}$	(1.49$^{0.5}$) 1.20$^{0.6}$

Table 5.16 – MAE when only using the cameras looking at outgoing traffic. The source models were trained on two cameras and the remaining one is used as target domain. The correlation coefficients between the real and predicted data are given as exponent to the MAE values. We report the accuracies without oracle information on top of the new ones (in between parenthesis). A green cell means that the results improve on both the source accuracy and the one obtained from the vanilla adaptation, an orange cell means that the results only improve on one of the two values.

	Source Average		Target *E73.268Y*$_4$		Target *E72.722Y*$_2$	
Regression-based:						
DeepMotionCLS	**0.80**$^{0.9}$	(0.83$^{0.9}$) **0.83**$^{0.9}$	4.78$^{0.8}$	(2.98$^{0.8}$) 1.34$^{0.9}$	2.90$^{0.4}$	(2.30$^{0.7}$) 1.13$^{0.8}$
DeepMotion3D	0.98$^{0.9}$	(1.01$^{0.9}$) 1.12$^{0.9}$	**4.40**$^{0.6}$	(3.06$^{0.7}$) **1.28**$^{0.9}$	**1.39**$^{0.6}$	(2.18$^{0.8}$) **0.77**$^{0.9}$
CTC-based:						
DeepMotion3D-CTC	-	-	-	-	-	-

Table 5.17 – Adaptation errors (MAE) and correlation coefficients between the real and predicted values for the two cameras looking at incoming traffic. Grayed columns are the errors before adaptation. We report the accuracies without oracle information on top of the new ones (in between parenthesis). A green cell means that the results improve on both the source accuracy and the one obtained from the vanilla adaptation, an orange cell means that the results only improve on one of the two values and a red cell means that the results are worse for the two values. Adaptation is rendered difficult as one of the camera is placed outside and due to the large difference in lane number between the cameras.

the fundamental diagrams of traffic flow theory (Figure 5.3), first growing linearly while in free flow and then decreasing when the traffic is jammed. As such, two solutions leveraging this information might be proposed to estimate the flow rate values *a priori*. The first solution consists in sampling inputs at various file size and annotating the selected files. Then, given a datapoint with a given file size, the associated flow rate value can be interpolated from close previously sampled inputs. The second solution consists in ensuring that data are equally distributed in the source and target domain along the various video file sizes, to equalize the densities of each flow rate value.

5.4 Conclusion

Gathering data is a tedious and lengthy process. While, for classical image processing, various datasets are publicly released ([Everingham et al., 2010, Lin et al., 2014]), for applications such as traffic flow estimation, data are much more scarce. For the purpose of traffic flow estimation we collected videos from surveillance cameras from Paris' tunnels. The dataset regroups recordings from five cameras for a total of 59 hours over 2 days along with the vehicles counting produced by induction loops every 20s.

Based on that, we explore the estimation of traffic flow rate using the compressed MPEG4

part-2 representation of the videos. For the sake, we design lightweight deep architectures that take as input MV frames either as sequences or as tensors. We show impressive speed gains when compared with a classical RGB detection-tracking-estimation method, while improving the accuracy. However, as data are still scarce, the trained networks barely generalize from one camera to another. Therefore, we apply domain adaptation to improve the estimation quality. Nevertheless, the domain adaptation results are not consistent enough across cameras to provide with a system that could be readily applied in real life conditions. Finally, pushing the analysis further, we experimentally show that the difficulties of adaptation are linked to changes between the source and the target domain output distributions. We empirically demonstrate the validity of such hypothesis, using oracle information to reduce the discrepancies between the source and target distributions during the adaptation process.

To conclude, the current work shows that the video compressed representation in a deep learning framework can be used for direct estimation of traffic flow parameters. However, multiple lines of research could be explored to try to improve the present work. To improve on the obtained results, annotation of more a diverse and larger dataset is paramount so as to consolidate and robustify the quality of the estimations. Furthermore, only the estimation of q is studied. However, we also collected the occupancy values. Estimation of such values is an interesting challenge that may require to modify the proposed approach. Indeed, as the o values soar when the vehicles are immobile, the MVs might not be sufficient for proper estimation. Two interesting lines of work to this problem are to add residual information or to consider subsequent datapoints dependent from one another.

Conclusion and Perspectives

Conlusion

We have addressed the challenge of learning deep vision networks using compressed images/videos. Specifically, we have tackled object detection in JPEG images and object counting from compressed MPEG4 part-2 videos, two tasks paramount to Actemium, a tech company in road surveillance. The main goal is to attain fast and lightweight networks which training departs from the use of classical RGB image/video representations, and which inference time of memory requirements may be suitable for real time surveillance of road traffic.

In chapter 3, we propose object detection networks using DCT coefficients issued from the JPEG representation as input. We demonstrate that a *frequency* representation can be used to carry out a *spatial* task. While carrying object detection in such space may seem counterintuitive, the tiled representation of the JPEG compression can be leveraged by the detection networks to locate the objects within the compressed images. Moreover, we empirically demonstrate that only part of the image information is required to produce accurate detections. Indeed, half of the DCT coefficients (in the high frequencies), as well as the color information, can be dropped without affecting on the detection accuracy. This lead to detection models as accurate as their RGB counterpart but $\times 1.7$ faster. These results are shown to be consistent across multiple datasets, both academical (Pascal VOC [Everingham et al., 2010] and MS-COCO [Lin et al., 2014]) and industrial (road tunnel images). These results tend to indicate that the vast majority of the object detection literature might be based on an over-informative input representation (RGB).

Chapters 4 and 5 tackle the task of flow rate estimation from compressed MPEG4 part-2 videos. The task is reminiscent of object counting in videos using solely motion vectors that characterize moving objects across video frames. The main issues are related to varying object scales and camera settings (orientation, illumination, occlusion, etc.). For the sake, we contribute two datasets: a synthetic one build upon MNIST and a real world dataset collected from 5 road tunnels' cameras during two days. Based on these datasets, we demonstrate the possibility to estimate the flow rate using lightweight deep neural networks. We show speed up gains up to $\times 3200$, while improving the accuracy, when compared with a classical detection-tracking-estimation method. However, we also highlight limitations in regard of the generalization capabilities of the proposed deep models. Because of variability in the camera settings, such a fact may hinder the effectiveness of the networks when deployed for road surveillance. To circumvent this problem, we investigate adaptation of the learned deep models to unannotated cameras, show the limitations of the method and demonstrate, using oracle information, that such limitations are due to discrepancies between the output distributions of the source and target domains.

In the thesis, we highlight the numerous advantages of rethinking existing methods towards a usage based on the compressed representation of data. In particular, we show the advantages of such approach for both image (object detection) and video (flow rate estimation) processing. We display impressive speed gains and bandwidth reduction while maintaining satisfactory accuracy. This approach, based on data pruning, is opposite to the mainstream deep learning way of tackling problems. Deep learning allows to learn the feature extraction

step, and, consequently, many have been considering the input fixed and have focused their attention on the models. However, through the experiments carried out in the thesis, we demonstrate twice over that impressive gains can be attained by trimming off unnecessary information. Such findings echo with a recent push from a part of the deep learning community to consider a more data-centric approach[4]. As some consider improving the datasets to challenge and improve the accuracy of the existing architectures, another approach would be to consider rethinking the inputs themselves to lighten the overall computation costs.

Perspectives

In chapter 3, we have empirically demonstrated that images contain superfluous information. Notably, the C_b and C_r channels, as well as the high frequency components in the DCT representation, are not required to provide with accurate object detections. However, during the experiments, we manually removed the high frequency coefficients at detection time. It would be more beneficial to learn the filtering operation during the training stage. This would allow to provide with the optimal number of DCT coefficients to remove, in order to minimize the bandwidth requirements while keeping the best accuracy. Such filter could be implemented using gates (e.g. similar to the ones used in ConvLSTM [Shi et al., 2015]) and weights regularization on gates linked with high frequency inputs.

In chapter 4 and 5, we have tackled the problem of flow rate estimation from compressed MPEG4 part-2 videos recording. Looking at the literature, it is obvious that the lack of reference datasets have been stalling the development of such new methods. We believe that the *first and foremost* task to be carried is the creation of a large reference dataset, so as to provide with a unified benchmark between the existing flow estimation methods. Although we come up with an interesting real-life dataset, the amount of collected data is far from enough to thoroughly evaluate proposed solutions. As diversity in data is paramount to deep learning, new annotation campaigns should be considered. Notably, a peculiar focus should be brought to increasing the number of available cameras, as well as the number of available hours per cameras.

Regarding the usage of compressed video representation for flow estimation, while we have addressed the usage of MV frames, we do not use the information available in the residual frames. However, such information has been used to filter out unreliable MVs [Yu et al., 2006]. The intuition being that vehicles' coefficients distribution differs from the road one, therefore providing with an easy way to remove MVs not tied to a vehicle. As such, combining MVs with residual frames may likely help the network focus on relevant information. Moreover, Yu et al. [2006] also show that only a few of the low frequency coefficients are required to perform accurate vehicle/road binary classification. Therefore, there is a huge potential for providing solutions that are fast, require few computation resources and little bandwidth, by considering adding only these coefficients to the MV inputs.

Finally, regarding the CTC-based architectures, even if we did not succeed to make them efficient in all configurations, promising results for some setups make us believe that the approach may yield great potential. One of the main limitation faced was the fixed number of outputs. Looking at the recent literature on image processing, such issue could be solved using recent and efficient models such as transformers. For instance, Carion et al. [2020] propose an object detection network that can output a varying number of objects, not tied *a priori* to a specific position. Adapting such architecture to output a varying number of flows would allow to solve the issue. If successful, such method for flow analysis could potentially be applied on a broad set of flow related problems, where exact annotation is tedious and difficult to obtain.

In the end, given the work proposed in the thesis as well as the perspectives, we trust that using the compressed representation of the data can provide with considerable resources

[4] https://https-deeplearning-ai.github.io/data-centric-comp/

savings. In a context where global warming is an ever more pressing issue, we hope that such approaches can become useful tools for the drastic energy consumption reduction challenges that are to come.

Appendix A

CTC: Computation of the forward and backward variables

We seek to compute the two variables α and β used in the computation of the CTC loss. Only the computation of the α variable is visually explained as the same underlying logic is applied to compute β. Formally, given the mapping $\mathcal{B}$ (as reminder: $\mathcal{B}(-TOO-O-) = \mathcal{B}(-T-O-OO) = TOO$), α is recursively computed as:

$$\alpha_t(s) = \begin{cases} (\alpha_{t-1}(s) + \alpha_{t-1}(s-1))y^t_{\mathrm{l'}_s} & \text{if } \mathrm{l'}_s = \mathit{blank} \text{ or } \mathrm{l'}_{s-2} = \mathrm{l'}_s \\ (\alpha_{t-1}(s) + \alpha_{t-1}(s-1) + \alpha_{t-1}(s-2))y^t_{\mathrm{l'}_s} & \text{otherwise.} \end{cases} \tag{A.1}$$

In equation Equation A.1, color matching Figure A.1 were used for better understanding.

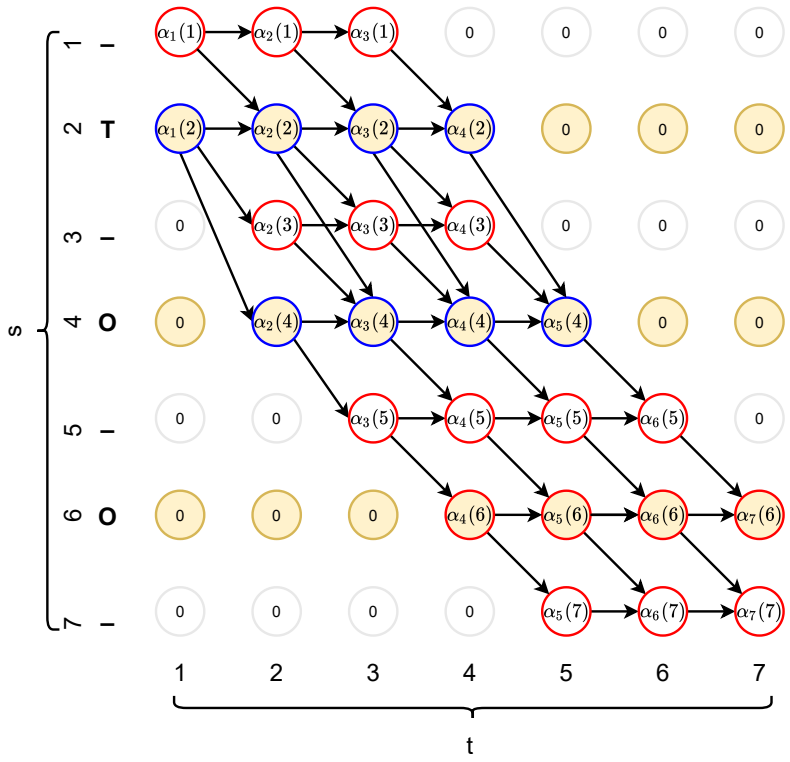


Figure A.1 – Representation of all possible forward paths. Colors were used to highlight the two different computation of nodes.

Noticing that there are only two possible starts, we get the following initialization:

$$\begin{aligned} \alpha_1(1) &= y_b^1, \\ \alpha_1(2) &= y_{\mathbf{l'}_1}^1, \\ \alpha_1(s) &= 0, \forall s > 2. \end{aligned} \tag{A.2}$$

Note that probabilities for symbols impossible to reach with a valid path are set to 0. Looking at the end of the figure, we see that the probability of $\mathbf{l}$ can be computed as:

$$p(\mathbf{l}|\mathbf{x}) = \alpha_T(|\mathbf{l'}|) + \alpha_T(|\mathbf{l'}| - 1). \tag{A.3}$$

Finally, we further introduce the backward variable β as:

$$\begin{aligned} \beta_T(|\mathbf{l'}|) &= y_b^T \\ \beta_T(|\mathbf{l'}| - 1) &= y_{\mathbf{l'}_{|\mathbf{l'}|}}^T \\ \beta_T(s) &= 0, \forall s < |\mathbf{l'}| - 1 \end{aligned} \tag{A.4}$$

$$\beta_t = \begin{cases} (\beta_{t+1}(s) + \beta_{t+1}(s+1))y_{\mathbf{l'}_s}^t & \text{if } \mathbf{l'}_s = blank \text{ or } \mathbf{l'}_{s-2} = \mathbf{l'}_s \\ (\beta_{t+1}(s) + \beta_{t+1}(s+1) + \beta_{t+1}(s+2))y_{\mathbf{l'}_s}^t & \text{otherwise} \end{cases} \tag{A.5}$$

Appendix B

Object detection in JPEG images

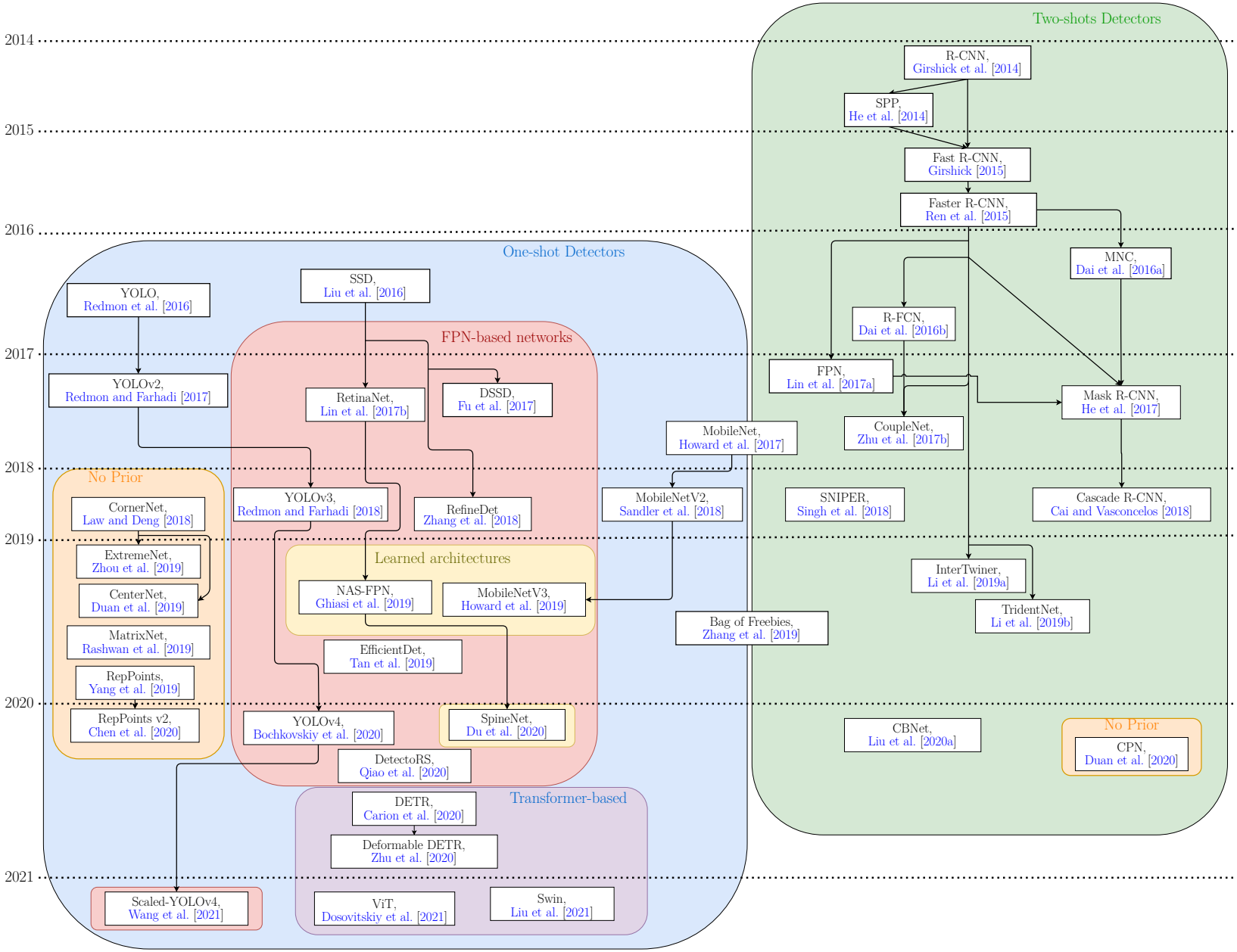


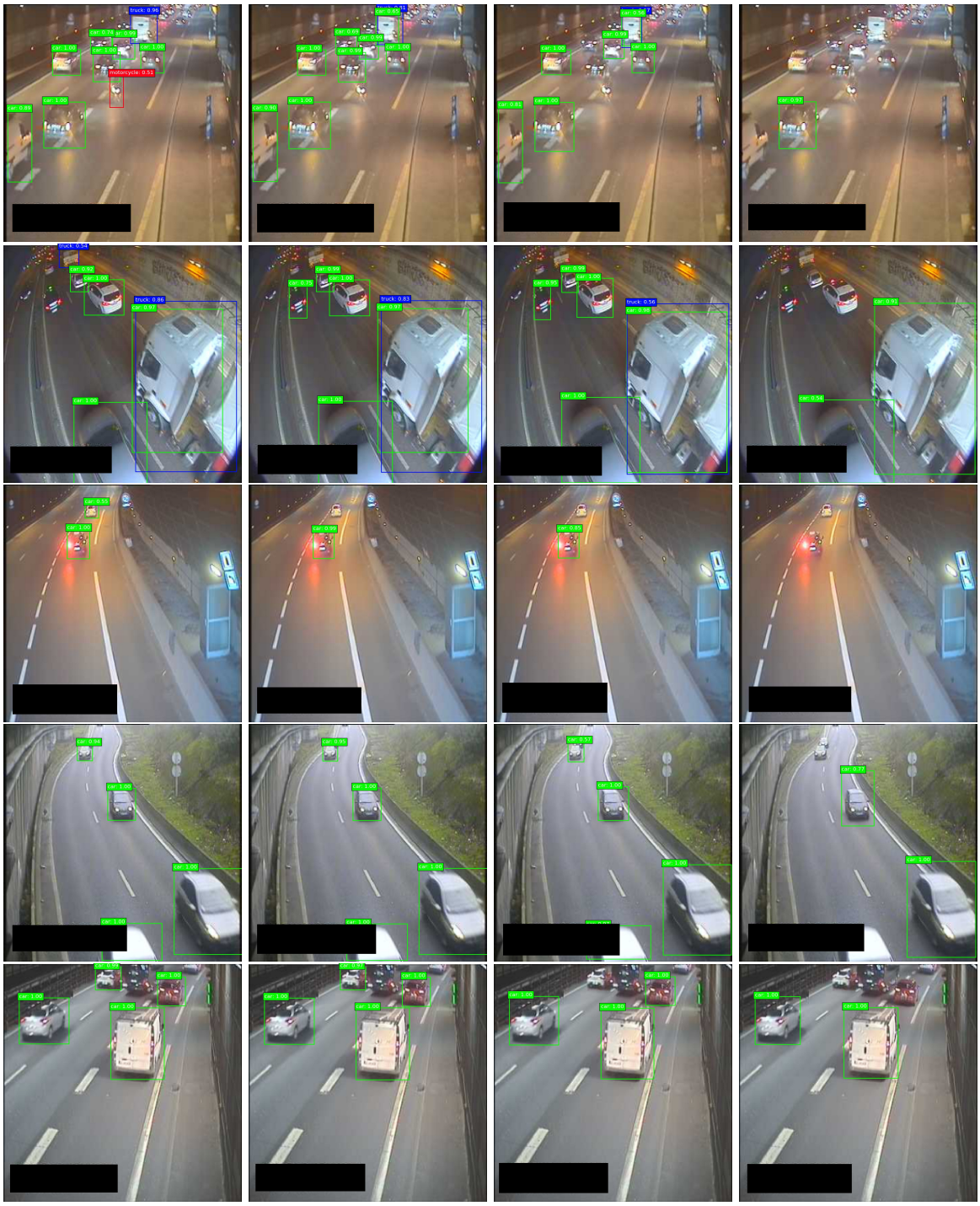
Figure B.1 – Evolution of the detection architectures through time.

Table B.1 – Features of the proposed detection networks. A * indicates that the layer is used for boxes prediction. The lines are arbitrary and do not represent the shape of the layers. Note that LC-RFA-thinner is skipped as it is a variation of the LC-RFA model.

SSD Liu et al. [2016]	SSD DCT (reference: SSD)		SSD DCT RFA (reference: SSD)		SSD DCT-deconv (reference: SSD)		SSD ResNet (reference: SSD)	SSD LC-RFA (reference: SSD ResNet)		SSD Deconvolution-RFA (reference SSD ResNet)	
RGB *(300,300,3)*	Y *(38,38,64)*	Cb, Cr *(19,19,128)*	Y *(38,38,64)*	Cb, Cr *(19,19,128)*	Y *(38,38,64)*	Cb, Cr *(19,19,128)*	RGB *(300,300,3)*	Y *(38,38,64)*	Cb, Cr *(19,19,128)*	Y *(38,38,64)*	Cb, Cr *(19,19,128)*
C_{11}, C_{12} P_1							C(64,7,2)				
C_{21}, C_{22} P_2							BN, R M(3,2)			Concat(39,39,192)	Deconv(28, 28, 128) - - - -
C_{31}, C_{32}, C_{33} P_3	BN, C(256,3,1)		BN C(324,3,1), C(324,3,1), C(324,3,1)		Concat(38,38,192)	Deconv(38,38,128) - - - -	CB_2(s=1) IB, IB	BN,CB_4(k=1,s=1) IB(k=2), IB		CB_4(k=1,s=1) IB(k=2), IB	
C_{41}, C_{42}, C^*_{43} P_4	C_{41}, C_{42}, C^*_{43} P_4 Concat	BN - - - -	C(324,3,1), C(324,3,1), C(324,3,1)* P_4 Concat	BN - - - -	BN, C_{41}, C_{42}, C^*_{43} P_4		CB_3 IB, IB, IB*	CB_3 IB, IB, IB* CB_3	BN, CB_3(k=1, s=1)	←	
C_{51}, C_{52}, C_{53} P_5 fc6, fc7* C_{61}, C^*_{62}	←		←		←		CB_4 IB, IB, IB, IB, IB CB_5(s=1), IB, IB, C_{61} C^*_{62}	Concat ←	- - - -		
C_{71} C^*_{72}							←				
C_{81}, C^*_{82}											
C_{91}, C^*_{92}											

Legend

RGB	RGB pixel input	Deconv	Deconvolution with 64 output channels, filter size 2, stride 2. Separate deconvolution layers are applied to Cb and to Cr, resulting in 128 total output channels
Y	Y channel DCT input	BN	BatchNormalization
Cb, Cr	Cb and Cr channel DCT input	R	Relu
C_{ij}	J-th convolution layer of the I-th block in the original SSD architecture	CB_n	ConvBlock stage n, with number of channels as in original ResNet-50 paper, kernel size = 3 and stride = 2 unless specified otherwise.
P_i	Pooling layer of the I-th block in the original SSD architecture	IB	IdentityBlock, with number of channels matched to preceding CB layer (as in ResNet-50)
C	Convolution(channels, filter size, stride)	M	MaxPooling(pool size, stride)
Concat	Channel wise concatenation	←	Layers after this point are the same as reference
- - - -	Channel is being concatenated	*	Layer is used for boxes prediction



(a) 64 coefficients kept (b) 32 coefficients kept (c) 16 coefficients kept (d) 8 coefficients kept

Figure B.2 – Detection performances (SSD DCT) on Actemium dataset depending on the number of DCT coefficients kept.

Appendix C

Flow rate estimation: Moving Digits

Network	Source Angle	0	45	90	135	180	225	270	315
DeepMotionCLF	0	**0.13**	3.14	3.66	1.45	**0.68**	2.44	3.21	1.91
	45	2.56	**0.17**	3.60	3.03	*0.70**	*0.56**	3.25	3.53
	90	3.71	3.29	**0.24**	3.43	3.71	3.31	*0.51**	3.52
	135	2.86	3.36	3.19	**0.18**	2.18	3.15	2.95	*1.24**
	180	*1.45**	1.53	1.57	1.57	1.47	1.51	1.50	1.49
	225	3.03	*1.40**	2.99	2.91	2.58	**0.18**	3.22	3.07
	270	2.37	2.89	*0.45**	3.10	1.95	2.79	**0.17**	3.02
	315	3.62	4.03	3.76	*0.83**	3.01	3.53	3.41	**0.19**
DeepMotionCLS	0	**0.10**	3.08	2.62	2.26	1.14	2.92	2.61	2.72
	45	3.40	**0.19**	3.07	3.51	1.29	*0.57**	2.73	3.89
	90	3.90	3.71	**0.22**	2.16	3.69	3.27	*0.55**	2.50
	135	3.44	4.09	3.53	**0.21**	2.40	3.21	3.28	*1.77**
	180	*2.05**	2.72	3.30	3.00	**0.22**	2.52	3.18	3.07
	225	2.72	*1.80**	3.12	3.18	1.62	**0.19**	3.20	3.38
	270	2.46	2.28	*0.75**	1.92	2.09	2.33	**0.10**	2.45
	315	2.93	3.93	3.12	*0.95**	*0.70**	2.98	2.53	**0.15**
DeepMotion3D	0	**0.19**	2.49	2.55	2.74	*0.45**	2.11	2.40	2.29
	45	2.26	**0.20**	1.86	*1.13**	0.99	1.47	2.28	2.91
	90	*1.70**	*1.68**	**0.25**	2.01	1.34	2.13	*0.76**	2.41
	135	2.95	2.97	2.84	**0.19**	0.79	2.09	2.75	1.99
	180	2.16	2.60	2.27	2.22	**0.23**	1.95	2.21	2.64
	225	2.53	2.22	2.31	1.87	0.93	**0.20**	2.07	2.84
	270	2.42	2.61	*0.53**	2.20	2.16	2.35	**0.19**	*1.88**
	315	2.21	3.38	2.15	2.75	1.33	*1.09**	1.21	**0.16**
DM3DCLF-CTC	0	**0.24**	4.35	*2.03**	**1.74**	1.02	4.33	**2.69**	*1.46**
	45	*0.38**	**0.21**	**0.96**	*2.44**	*0.74**	*2.63**	4.68	**0.86**
	90	3.42	3.38	3.40	3.32	3.42	3.40	*3.28**	3.37
	135	3.42	3.38	3.40	3.32	3.42	3.40	*3.28**	3.37
	180	0.66	4.31	4.38	2.96	**0.16**	4.28	4.26	3.66
	225	1.36	*0.85**	8.06	7.34	2.27	**0.33**	7.01	6.30
	270	3.42	3.38	3.40	3.32	3.42	3.40	*3.28**	3.37
	315	3.42	3.38	3.40	3.32	3.42	3.40	*3.28**	3.37
DeepMotion3D-CTC	0	**0.04**	4.37	4.38	4.30	**0.87**	4.39	4.26	4.36
	45	*0.59**	**0.14**	1.04	*0.79**	*1.08**	**0.84**	*1.24**	**0.80**
	90	4.40	4.35	**0.06**	4.24	4.41	*3.34**	1.34	4.29
	135	3.24	4.37	4.38	**0.08**	4.27	4.39	4.26	*0.82**
	180	3.01	*3.38**	3.40	3.29	2.89	3.39	3.28	3.34
	225	3.42	*3.38**	3.40	3.32	3.42	3.40	3.28	3.37
	270	4.40	4.37	*0.89**	4.30	4.41	4.39	**0.02**	4.36
	315	3.42	*3.38**	3.40	3.32	3.42	3.40	3.28	3.37

Table C.1 – Accuracy for the various orientation trainings. Networks are trained on a single orientation and tested on all remaining others. Bold font refers to the best accuracy from the column and italic starred to the second best. Higher errors are highlighted using a more vivid orange.

Appendix D

Traffic flow parameters estimation: Actemium Dataset

	Source	Target				
		E72.433F$_3$	*E72.722Y$_2$*	*E73.268Y$_4$*	E73.332A$_4$	E73.531M$_3$
DeepMCLS	E72.433F	**0.57**$^{1.0}$	3.91$^{0.3}$	2.83$^{0.1}$	2.15$^{-0.1}$	2.34$^{0.5}$
	E72.722Y	1.16$^{0.9}$	**0.91**$^{0.9}$	2.90$^{0.4}$	2.73$^{-0.2}$	3.18$^{0.7}$
	E73.268Y	2.10$^{0.9}$	4.78$^{0.8}$	**0.68**$^{0.9}$	2.01$^{0.3}$	3.27$^{0.5}$
	E73.332A	2.66$^{0.8}$	6.07$^{0.2}$	2.62$^{0.75}$	0.64$^{0.9}$	1.92$^{0.8}$
	E73.531M	3.09$^{0.8}$	6.52$^{0.3}$	4.73$^{0.3}$	**0.90**$^{0.8}$	**0.44**$^{1.0}$
DeepM3D	E72.433F	**0.63**$^{0.9}$	3.47$^{0.7}$	2.09$^{0.3}$	**1.33**$^{0.6}$	1.03$^{0.7}$
	E72.722Y	0.93$^{0.9}$	**1.12**$^{0.9}$	1.39$^{0.6}$	1.33$^{0.5}$	1.48$^{0.5}$
	E73.268Y	2.22$^{0.9}$	4.40$^{0.6}$	**0.83**$^{0.9}$	1.74$^{0.6}$	2.83$^{0.6}$
	E73.332A	2.80$^{0.8}$	5.67$^{0.25}$	1.36$^{0.6}$	0.74$^{0.8}$	1.77$^{0.8}$
	E73.531M	2.83$^{0.8}$	5.22$^{0.4}$	2.00$^{0.4}$	2.24$^{0.8}$	**0.56**$^{0.9}$
DeepM3D-CTC	E72.433F	**0.75**$^{0.9}$	-	-	-	2.15$^{0.4}$
	E72.722Y	-	**1.03**$^{0.9}$	-	-	-
	E73.268Y	-	-	**1.93**$^{0.7}$	2.00$^{0.0}$	-
	E73.332A	-	-	5.43$^{-0.1}$	**1.91**$^{0.7}$	-
	E73.531M	3.67$^{0.4}$	-	-	-	**0.51**$^{0.9}$

Table D.1 – Generalization performances (MAE and correlation coefficient) for the networks trained on each camera separately. For clarity of the notations, correlation coefficients are provided as exponent. Results on other target cameras are obtained without fine-tuning. Bold results are the best results for a given architecture. Higher errors are highlighted using a more vivid orange. Due to the limitations of the CTC-based architecture, only the camera with a similar number of lanes as the source are considered.

Algorithm 1 Estimation of q values

```
1: function EstimateQT(nol, threshold)                    # numberOfLanes
2:     currentTracks ← List()
3:     lostTracks ← List()
4:     # First two modules (detection and tracking)
5:     for frame in video do
6:         detections ← ObjectDetection(frame)
7:         UpdateTracks(currentTracks, lostTracks, detections)
8:     end for
9:     for track in currentTracks do
10:        lostTracks.add(track)
11:    end for
12:    # Post-Processing
13:    ClearTracks(lostTracks)
14:    # Estimate q
15:    for track in lostTracks do
16:        if track.length < threshold then
17:            GoTo next track
18:        end if
19:        if CrossLoop(loopPosition, track) then
20:            Q ← Q + 1
21:        end if
22:    end for
23:    Q ← Q/nol
24: end function
```

Algorithm 2 Update the active tracks and save the finished ones

```
1:  function UpdateTracks(currentTracks, lostTracks, detections)
2:      if detections.length == 0 then
3:          for track in currentTracks do                          # Update
4:              track.skippedFrames ← track.skippedFrames + 1
5:              track.objectBBox.append(None)
6:          end for
7:          for track in currentTracks do                          # Clean
8:              if track.skippedFrames > maxFrameSkipped then
9:                  processedTracks.append(track)
10:                 currentTracks.remove(track)
11:             end if
12:         end for
          return
13:     end if
14:     if currentTracks.length == 0 then
15:         for detection in detections do                         # Update
16:             track ← Track(detection)
17:             currentTracks.append(track)
18:         end for
          return
19:     end if
20:     cost ← centerCost(currentTracks, detections)
21:     assignments ← linearSumAssignment(cost)
22:     detectionAssigned ← List(False, detections.length)
23:     trackletAssigned ← List(False, currentTracks.length)
24:     for row, col in assignments do
25:         if cost[row, col] <= costThreshold then
26:             currentTracks[row].update(detections[col])
27:             detectionAssigned[col] ← True
28:             trackletAssigned[row] ← True
29:         end if
30:     end for
```

```
    for i = 1 to trackletAssigned.length do
        if nottrackletAssigned[i] then
            currentTracks[i].skippedFrames+ = 1
            currentTracks[i].objectBBox.append(None)
        end if
    end for
    for track in currentTracks do
        if track.skippedFrames > maxFrameSkipped then
            processedTracks.append(track)
            currentTracks.remove(track)
        end if
    end for
    for i = 1 to detectionAssigned.length do
        if notdetectionAssigned[i] then
            newTrack ← Track(detections[i])
            currentTracks.append(newTrack)
        end if
    end for
end function
```

Algorithm 3 Tells if a tracks crosses one of the lanes

```
function CrossLoop(loopPosition, track)
    for i = 1 to track.length − 1 do
        trackPosition ← [track.position[i], track.position[i + 1]]
        if intersects(loopPosition, trackPosition) then
            return True
        end if
    end for
    return False
end function
```

Bibliography

Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, Jie Chen, Jingdong Chen, Zhijie Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Ke Ding, Niandong Du, Erich Elsen, Jesse Engel, Weiwei Fang, Linxi Fan, Christopher Fougner, Liang Gao, Caixia Gong, Awni Hannun, Tony Han, Lappi Johannes, Bing Jiang, Cai Ju, Billy Jun, Patrick LeGresley, Libby Lin, Junjie Liu, Yang Liu, Weigao Li, Xiangang Li, Dongpeng Ma, Sharan Narang, Andrew Ng, Sherjil Ozair, Yiping Peng, Ryan Prenger, Sheng Qian, Zongfeng Quan, Jonathan Raiman, Vinay Rao, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Kavya Srinet, Anuroop Sriram, Haiyuan Tang, Liliang Tang, Chong Wang, Jidong Wang, Kaifu Wang, Yi Wang, Zhijian Wang, Zhiqian Wang, Shuang Wu, Likai Wei, Bo Xiao, Wen Xie, Yan Xie, Dani Yogatama, Bin Yuan, Jun Zhan, and Zhenyao Zhu. Deep speech 2 : End-to-end speech recognition in english and mandarin. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 173–182, New York, New York, USA, 20–22 Jun 2016. PMLR. URL http://proceedings.mlr.press/v48/amodei16.html.

Philipp Bergmann, Tim Meinhardt, and Laura Leal-Taixé. Tracking without bells and whistles. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

Marcin Bernaś. Objects detection and tracking in highly congested traffic using compressed video sequences. In Leonard Bolc, Ryszard Tadeusiewicz, Leszek J. Chmielewski, and Konrad Wojciechowski, editors, *Computer Vision and Graphics*, pages 296–303, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-33564-8.

Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 3464–3468, 2016. doi: 10.1109/ICIP.2016.7533003.

Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *CoRR*, abs/2004.10934, 2020. URL https://arxiv.org/abs/2004.10934.

Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S. Davis. Soft-nms - improving object detection with one line of code. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 5562–5570. IEEE, 2017. ISBN 978-1-5386-1032-9. doi: 10.1109/ICCV.2017.593. URL http://doi.ieeecomputersociety.org/10.1109/ICCV.2017.593.

Ivan Brkić, Mario Miler, Marko Ševrović, and Damir Medak. An analytical framework for accurate traffic flow parameter calculation from uav aerial videos. *Remote Sensing*, 12 (22), 2020. ISSN 2072-4292. doi: 10.3390/rs12223844. URL https://www.mdpi.com/2072-4292/12/22/3844.

Nam Bui, Hongsuk Yi, and Jiho Cho. A vehicle counts by class framework using distinguished regions tracking at multiple intersections. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020.

Zhaowei Cai and N. Vasconcelos. Cascade r-cnn: Delving into high quality object detection. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6154–6162, 2018.

F. W. Campbell and J. G. Robson. Application of fourier analysis to the visibility of gratings. *J Physiol*, 197(3):551–566, August 1968.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part I*, volume 12346 of *Lecture Notes in Computer Science*, pages 213–229. Springer, 2020. doi: 10.1007/978-3-030-58452-8_13. URL https://doi.org/10.1007/978-3-030-58452-8_13.

L D. Chamain and Z Ding. Faster and accurate classification for jpeg2000 compressed images in networked applications. *ArXiv*, abs/1909.05638, 2019.

M. Chang, C. Chiang, C. Tsai, Y. Chang, H. Chiang, Y. Wang, S. Chang, Y. Li, M. Tsai, and H. Tseng. Ai city challenge 2020 – computer vision for smart transportation applications. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2638–2647, 2020. doi: 10.1109/CVPRW50498.2020.00318.

W. Chen, C. Smith, and S. Fralick. A fast computational algorithm for the discrete cosine transform. *IEEE Trans. Commun.*, 25:1004–1009, 1977.

Yihong Chen, Zheng Zhang, Yue Cao, Liwei Wang, Stephen Lin, and Han Hu. Reppoints v2: Verification meets regression for object detection. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/3ce3bd7d63a2c9c81983cc8e9bd02ae5-Abstract.html.

Y. Cho and J. Rice. Estimating velocity fields on a freeway from low-resolution videos. *IEEE Transactions on Intelligent Transportation Systems*, 7(4):463–469, Dec 2006. ISSN 1524-9050. doi: 10.1109/TITS.2006.883934.

Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper/2015/file/1068c6e4c8051cfd4e9ea8072e3189e2-Paper.pdf.

Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Jifeng Dai, Kaiming He, and Jian Sun. Instance-aware semantic segmentation via multi-task network cascades. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3150–3158, 2016a.

Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *NIPS*, pages 379–387, 2016b. URL http://dblp.uni-trier.de/db/conf/nips/nips2016.html#DaiLHS16.

Bharath B. Damodaran, Benjamin Kellenberger, Rémi Flamary, Devis Tuia, and Nicolas Courty. Deepjdot: Deep joint distribution optimal transport for unsupervised domain adaptation. In *European Conference in Computer Visions (ECCV)*, 2018.

Benjamin Deguerre, Clément Chatelain, and Gilles Gasso. Fast object detection in compressed jpeg images. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 333–338, 2019. doi: 10.1109/ITSC.2019.8916937.

Benjamin Deguerre, Clement Chatelain, and Gilles Gasso. Object detection in the dct domain: is luminance the solution? In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 2627–2634, 2021. doi: 10.1109/ICPR48806.2021.9412998.

Samuel Felipe dos Santos and Jurandy Almeida. Deep learning towards edge computing: Neural networks straight from compressed data. *CoRR*, abs/2012.14426, 2020. URL https://arxiv.org/abs/2012.14426.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*, 2021.

Xianzhi Du, Tsung-Yi Lin, Pengchong Jin, Golnaz Ghiasi, Mingxing Tan, Yin Cui, Quoc V. Le, and Xiaodan Song. Spinenet: Learning scale-permuted backbone for recognition and localization. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11589–11598, 2020.

Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. Centernet: Keypoint triplets for object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

Kaiwen Duan, Lingxi Xie, Honggang Qi, Song Bai, Qingming Huang, and Qi Tian. Corner proposal network for anchor-free, two-stage object detection. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, pages 399–416, Cham, 2020. Springer International Publishing. ISBN 978-3-030-58580-8.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

Adam Dziedzic, John Paparrizos, Sanjay Krishnan, Aaron Elmore, and Michael Franklin. Band-limited training and inference for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1745–1754. PMLR, 09–15 Jun 2019. URL http://proceedings.mlr.press/v97/dziedzic19a.html.

Max Ehrlich and Larry S. Davis. Deep residual learning in the jpeg transform domain. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2): 303–338, June 2010.

R. M. Fano. The transmission of information. Technical Report 65, Research Laboratory for Electronics, MIT, Cambridge, MA, USA, 1949.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2021.

Cheng-Yang Fu, Wei Liu, Ananth Ranga, Ambrish Tyagi, and Alexander C. Berg. DSSD : Deconvolutional single shot detector. *CoRR*, abs/1701.06659, 2017. URL http://arxiv.org/abs/1701.06659.

Yuan Hu Fu, Hichem Sahli, Xing Fa Dong, and Jian Wang. A high efficient system for traffic mean speed estimation from mpeg video. *Artificial Intelligence and Computational Intelligence, International Conference on*, 3:444–448, 01 2009. doi: 10.1109/AICI.2009.358.

Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

Felix A. Gers, Jürgen A. Schmidhuber, and Fred A. Cummins. Learning to forget: Continual prediction with lstm. *Neural Comput.*, 12(10):2451–2471, October 2000. ISSN 0899-7667. doi: 10.1162/089976600300015015. URL https://doi.org/10.1162/089976600300015015.

Golnaz Ghiasi, Tsung-Yi Lin, Ruoming Pang, and Quoc V. Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7029–7038, 2019.

R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 00, pages 580–587, June 2014. doi: 10.1109/CVPR.2014.81. URL https://ieeexplore.ieee.org/abstract/document/6909475/.

Ross Girshick. Fast r-cnn. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, page 1440–1448, USA, 2015. IEEE Computer Society. ISBN 9781467383912. doi: 10.1109/ICCV.2015.169. URL https://doi.org/10.1109/ICCV.2015.169.

Shreyank N. Gowda, Marcus Rohrbach, and Laura Sevilla-Lara. SMART frame selection for action recognition. *CoRR*, abs/2012.10671, 2020. URL https://arxiv.org/abs/2012.10671.

P Goyal, P Dollár, R B. Girshick, P Noordhuis, L Wesolowski, A Kyrola, A Tulloch, Y Jia, and K He. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017. URL http://arxiv.org/abs/1706.02677.

Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1764–1772, Bejing, China, 22–24 Jun 2014. PMLR. URL http://proceedings.mlr.press/v32/graves14.html.

Alex Graves, Santiago Fernández, and Faustino Gomez. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *In Proceedings of the International Conference on Machine Learning, ICML 2006*, pages 369–376, 2006.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, 2013. doi: 10.1109/ICASSP.2013.6638947.

Klaus Greff, Rupesh K. Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232, 2017. doi: 10.1109/TNNLS.2016.2582924.

L. Gueguen, A. Sergeev, B. Kadlec, R. Liu, and J. Yosinski. Faster neural networks straight from jpeg. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 3933–3944. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/7649-faster-neural-networks-straight-from-jpeg.pdf.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE CVPR*, pages 770–778, June 2016. doi: 10.1109/CVPR.2016.90.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 346–361, Cham, 2014. Springer International Publishing. ISBN 978-3-319-10578-9.

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017. doi: 10.1109/ICCV.2017.322.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. URL http://arxiv.org/abs/1704.04861.

Andrew Howard, Mark Sandler, Bo Chen, Weijun Wang, Liang-Chieh Chen, Mingxing Tan, Grace Chu, Vijay Vasudevan, Yukun Zhu, Ruoming Pang, Hartwig Adam, and Quoc Le. Searching for mobilenetv3. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1314–1324, 2019. doi: 10.1109/ICCV.2019.00140.

David Huffman. A method for the construction of minimum redundancy codes. *Proceedings of the Institute of Radio Engineers*, 40(9):1098–1101, 1952. ISSN 0096-8390. doi: https://doi.org/10.1109/jrproc.1952.273898.

T Ide, T Katsuki, T Morimura, and R Morris. City-wide traffic flow estimation from a limited number of low-quality cameras. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–10, 08 2016. doi: 10.1109/TITS.2016.2597160.

LH Immers and S Logghe. Traffic flow theory. *Faculty of Engineering, Department of Civil Engineering, Section Traffic and Infrastructure, Kasteelpark Arenberg*, 40(21), 2002.

Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35, 1960. doi: 10.1115/1.3662552. URL http://dx.doi.org/10.1115/1.3662552.

Christian Kas, Mathieu Brulin, Henri Nicolas, and Christophe Maillet. Compressed domain aided analysis of traffic surveillance videos. In *2009 Third ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC)*, pages 1–8, 2009. doi: 10.1109/ICDSC.2009.5289345.

Ruimin Ke, Zhibin Li, Sung Kim, John Ash, Zhiyong Cui, and Yinhai Wang. Real-time bidirectional traffic flow parameter estimation from aerial videos. *IEEE Transactions on Intelligent Transportation Systems*, 18(4):890–901, 2017. doi: 10.1109/TITS.2016.2595526.

Ruimin Ke, Zhibin Li, Jinjun Tang, Zewen Pan, and Yinhai Wang. Real-time traffic flow parameter estimation from uav video based on ensemble classifier and optical flow. *IEEE Transactions on Intelligent Transportation Systems*, 20(1):54–64, 2019. doi: 10.1109/TITS.2018.2797697.

Henry J Kelley. Gradient theory of optimal flight paths. *Ars Journal*, 30(10):947–954, 1960.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Serkan Kiranyaz, Onur Avci, Osama Abdeljaber, Turker Ince, Moncef Gabbouj, and Daniel J. Inman. 1d convolutional neural networks and applications: A survey. *Mechanical Systems and Signal Processing*, 151:107398, 2021. ISSN 0888-3270. doi: https://doi.org/10.1016/j.ymssp.2020.107398. URL https://www.sciencedirect.com/science/article/pii/S0888327020307846.

Robert Krajewski, Julian Bock, Laurent Kloeker, and Lutz Eckstein. The highd dataset: A drone dataset of naturalistic vehicle trajectories on german highways for validation of highly automated driving systems. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2118–2125, 2018. doi: 10.1109/ITSC.2018.8569552.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 and cifar-100 (canadian institute for advanced research). URL http://www.cs.toronto.edu/~kriz/cifar.html.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

H. W. Kuhn and Bryn Yaw. The hungarian method for the assignment problem. *Naval Res. Logist. Quart*, pages 83–97, 1955.

Amit Kumar, Pirazh Khorramshahi, Wei-An Lin, Prithviraj Dhar, Jun-Cheng Chen, and Rama Chellappa. A semi-automatic 2d solution for vehicle speed estimation from monocular videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018.

Ichraf Lahouli, Zied Chtourou, Mohamed Ali Ben Ayed, Robby Haelterman, Geert De Cubber, and Rabah Attia. Pedestrian detection and trajectory estimation in the compressed domain using thermal images. In Dominique Bechmann, Manuela Chessa, Ana Paula Cláudio, Francisco H. Imai, Andreas Kerren, Paul Richard, Alexandru C. Telea, and Alain Trémeau, editors, *Computer Vision, Imaging and Computer Graphics Theory and Applications - 13th International Joint Conference, VISIGRAPP 2018, Funchal, Madeira, Portugal, January 27-29, 2018, Revised Selected Papers*, volume 997 of *Communications in Computer and Information Science*, pages 212–227. Springer, 2018. doi: 10.1007/978-3-030-26756-8_10. URL https://doi.org/10.1007/978-3-030-26756-8_10.

Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

Gwang-Gook Lee, Byeoung-su Kim, and Whoi-Yul Kim. Automatic estimation of pedestrian flow. In *2007 First ACM/IEEE International Conference on Distributed Smart Cameras*, pages 291–296, 2007. doi: 10.1109/ICDSC.2007.4357536.

F. Li, Fatih Porikli, and Xiaokun Li. Traffic congestion estimation using hmm models without vehicle tracking. In *In IEEE Intelligent Vehicle Symposium*, pages 188–193, 2004.

Hongyang Li, Bo Dai, Shaoshuai Shi, Wanli Ouyang, and Xiaogang Wang. Feature intertwiner for object detection. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019a. URL https://openreview.net/forum?id=SyxZJn05YX.

Jinlong Li, Zhigang Xu, Lan Fu, Xuesong Zhou, and Hongkai Yu. Domain adaptation from daytime to nighttime: A situation-sensitive vehicle detection and traffic flow parameter estimation framework. *Transportation Research Part C: Emerging Technologies*, 124:102946, 2021. ISSN 0968-090X. doi: https://doi.org/10.1016/j.trc.2020.102946. URL https://www.sciencedirect.com/science/article/pii/S0968090X20308433.

Yanghao Li, Yuntao Chen, Naiyan Wang, and Zhao-Xiang Zhang. Scale-aware trident networks for object detection. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 6053–6062. IEEE, 2019b. doi: 10.1109/ICCV.2019.00615. URL https://doi.org/10.1109/ICCV.2019.00615.

Z. Li and F. Zhou. FSSD: feature fusion single shot multibox detector. *CoRR*, abs/1712.00960, 2017. URL http://arxiv.org/abs/1712.00960.

Tsung-Yi Lin, M. Maire, Serge J. Belongie, James Hays, P. Perona, D. Ramanan, Piotr Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014.

Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 936–944, 2017a. doi: 10.1109/CVPR.2017.106.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2999–3007, 2017b. doi: 10.1109/ICCV.2017.324.

Zachary Chase Lipton. A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019, 2015. URL http://arxiv.org/abs/1506.00019.

Christian J. van den Branden Lambrecht (eds.) Lisa J. Croner, Thomas Wachtler (auth.). *Vision Models and Applications to Image and Video Processing*. Springer US, 1 edition, 2001. ISBN 978-1-4419-4905-9,978-1-4757-3411-9.

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *ECCV (1)*, volume 9905 of *Lecture Notes in Computer Science*, pages 21–37. Springer, 2016. ISBN 978-3-319-46447-3. URL http://dblp.uni-trier.de/db/conf/eccv/eccv2016-1.html#LiuAESRFB16.

Yudong Liu, Yongtao Wang, Siwei Wang, Tingting Liang, Qijie Zhao, Zhi Tang, and Haibin Ling. Cbnet: A novel composite backbone network architecture for object detection. In *AAAI*, 2020a.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *CoRR*, abs/2103.14030, 2021. URL https://arxiv.org/abs/2103.14030.

Zhongji Liu, Wei Zhang, Xu Gao, Hao Meng, Xiao Tan, Xiaoxing Zhu, Zhan Xue, Xiaoqing Ye, Hongwu Zhang, Shilei Wen, and Errui Ding. Robust movement-specific vehicle counting at crowded intersections. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020b.

Shao-Yuan Lo and Hsueh-Ming Hang. Exploring semantic segmentation on the dct representation. In *Proceedings of the ACM Multimedia Asia*, MMAsia '19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450368414. doi: 10.1145/3338533.3366557. URL https://doi.org/10.1145/3338533.3366557.

C. Löffler, A. Ligtenberg, and G. Moschytz. Practical fast 1-d dct algorithms with 11 multiplications. *International Conference on Acoustics, Speech, and Signal Processing,*, pages 988–991 vol.2, 1989.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, 2015. doi: 10.1109/CVPR.2015.7298965.

David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004. ISSN 0920-5691. doi: 10.1023/B:VISI.0000029664.99615.94. URL http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94.

Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, page 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc.

K.P. Mbonye and F.P. Ferrie. Attentive visual servoing in the mpeg compressed domain for un-calibrated motion parameter estimation of road traffic. In *18th International Conference on Pattern Recognition (ICPR'06)*, volume 4, pages 908–911, 2006. doi: 10.1109/ICPR.2006.281.

Warren Mcculloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:127–147, 1943.

Gaspard Monge. *Mémoire sur la théorie des déblais et des remblais*. De l'Imprimerie Royale, 1781.

Milind Naphade, David C. Anastasiu, Anuj Sharma, Vamsi Jagrlamudi, Hyeran Jeon, Kaikai Liu, Ming-Ching Chang, Siwei Lyu, and Zeyu Gao. The nvidia ai city challenge. In *Prof. SmartWorld*, Santa Clara, CA, USA, 2017.

Milind Naphade, Ming-Ching Chang, Anuj Sharma, David C. Anastasiu, Vamsi Jagarlamudi, Pranamesh Chakraborty, Tingting Huang, Shuo Wang, Ming-Yu Liu, Rama Chellappa, Jenq-Neng Hwang, and Siwei Lyu. The 2018 nvidia ai city challenge. In *Proc. CVPR Workshops*, pages 53—60, 2018.

Milind Naphade, Zheng Tang, Ming-Ching Chang, David C. Anastasiu, Anuj Sharma, Rama Chellappa, Shuo Wang, Pranamesh Chakraborty, Tingting Huang, Jenq-Neng Hwang, and Siwei Lyu. The 2019 ai city challenge. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, page 452–460, June 2019.

Milind Naphade, Shuo Wang, David C. Anastasiu, Zheng Tang, Ming-Ching Chang, Xiaodong Yang, Liang Zheng, Anuj Sharma, Rama Chellappa, and Pranamesh Chakraborty. The 4th ai city challenge. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, page 2665–2674, June 2020.

Gabriel Peyré and Marco Cuturi. Computational optimal transport, 2020.

Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, 1999. URL http://dblp.uni-trier.de/db/journals/nn/nn12.html#Qian99.

Siyuan Qiao, Liang-Chieh Chen, and Alan L. Yuille. Detectors: Detecting objects with recursive feature pyramid and switchable atrous convolution. *CoRR*, abs/2006.02334, 2020. URL https://arxiv.org/abs/2006.02334.

Abdullah Rashwan, Agastya Kalra, and Pascal Poupart. Matrix nets: A new deep architecture for object detection. In *2019 IEEE/CVF International Conference on Computer Vision Workshops, ICCV Workshops 2019, Seoul, Korea (South), October 27-28, 2019*, pages 2025–2028. IEEE, 2019. doi: 10.1109/ICCVW.2019.00252. URL https://doi.org/10.1109/ICCVW.2019.00252.

Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6517–6525, 2017. doi: 10.1109/CVPR.2017.690.

Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018. URL http://arxiv.org/abs/1804.02767.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf.

F. Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. *American Journal of Psychology*, 76:705, 1963.

O Russakovsky, J Deng, H Su, J Krause, S Satheesh, S Ma, Z Huang, A Karpathy, A Khosla, M Bernstein, A C. Berg, and L Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018. doi: 10.1109/CVPR.2018.00474.

Samuel Felipe dos Santos, Nicu Sebe, and Jurandy Almeida. The good, the bad, and the ugly: Neural networks straight from jpeg. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 1896–1900, 2020. doi: 10.1109/ICIP40778.2020.9190741.

T. N. Schoepflin and D. J. Dailey. Algorithms for calibrating roadside traffic cameras and estimating mean vehicle speed. In *2007 IEEE Intelligent Transportation Systems Conference*, pages 277–283, 2007. doi: 10.1109/ITSC.2007.4357806.

Claude E. Shannon. A mathematical theory of communication. *Bell Syst. Tech. J.*, 27 (3):379–423, 1948. URL http://dblp.uni-trier.de/db/journals/bstj/bstj27.html#Shannon48.

H. Shi, Z. Wang, Y. Zhang, X. Wang, and T. Huang. Geometry-aware traffic flow analysis by detection and tracking. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 116–1164, 2018. doi: 10.1109/CVPRW.2018.00023.

Jianbo Shi and Carlo Tomasi. Good features to track. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600, 1994.

Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun WOO. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper/2015/file/07563a3fe3bbe7e3ba84431ad9d055af-Paper.pdf.

Z Shou, Z Yan, Y Kalantidis, L Sevilla-Lara, M Rohrbach, X Lin, and S-F Chang. Dmc-net: Generating discriminative motion cues for fast compressed video action recognition. *CoRR*, abs/1901.03460, 2019. URL http://arxiv.org/abs/1901.03460.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. URL http://arxiv.org/abs/1409.1556.

Bharat Singh, Mahyar Najibi, and Larry S Davis. Sniper: Efficient multi-scale training. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/166cee72e93a992007a89b39eb29628b-Paper.pdf.

Satya P. Singh, Lipo Wang, Sukrit Gupta, Haveesh Goli, Parasuraman Padmanabhan, and Balázs Gulyás. 3d deep learning on medical images: A review. *Sensors*, 20(18), 2020. ISSN 1424-8220. doi: 10.3390/s20185097. URL https://www.mdpi.com/1424-8220/20/18/5097.

Corey Snyder and Minh Do. Streets: A novel camera network dataset for traffic flow. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/ee389847678a3a9d1ce9e4ca69200d06-Paper.pdf.

Yaohang Sun, Zhen Liu, and Zhisong Pan. Intersection traffic flow counting based on hybrid regression model. In *2019 IEEE International Conference on Signal, Information and Data Processing (ICSIDP)*, pages 1–4, 2019. doi: 10.1109/ICSIDP47821.2019.9173285.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition,*, 2016. URL http://arxiv.org/abs/1512.00567.

Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 09–15 Jun 2019. URL http://proceedings.mlr.press/v97/tan19a.html.

Mingxing Tan, Ruoming Pang, and Quoc V. Le. Efficientdet: Scalable and efficient object detection. *CoRR*, abs/1911.09070, 2019. URL http://arxiv.org/abs/1911.09070.

Zheng Tang, Gaoang Wang, Hao Xiao, Aotian Zheng, and Jenq-Neng Hwang. Single-camera and inter-camera vehicle tracking and 3d speed estimation based on fusion of visual and semantic features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018.

T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

Carlo Tomasi and Takeo Kanade. Detection and tracking of point features. Technical report, International Journal of Computer Vision, 1991.

M. Tran, T. Dinh-Duy, T. Truong, V. Ton-That, T. Do, Q. Luong, T. Nguyen, V. Nguyen, and M. N. Do. Traffic flow analysis with multiple adaptive vehicle detectors and velocity estimation with landmark-based scanlines. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 100–1007, 2018. doi: 10.1109/CVPRW.2018.00021.

Roland Tusch, Felix Pletzer, Armin Krätschmer, Laszlo Böszörmenyi, Bernhard Rinner, Thomas Mariacher, and Manfred Harrer. Efficient level of service classification for traffic monitoring in the compressed video domain. In *2012 IEEE International Conference on Multimedia and Expo*, pages 967–972, 2012. doi: 10.1109/ICME.2012.101.

J.R.R. Uijlings, K.E.A. van de Sande, T. Gevers, and A.W.M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 2013. doi: 10.1007/s11263-013-0620-5. URL http://www.huppelen.nl/publications/selectiveSearchDraft.pdf.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008. URL http://www.jmlr.org/papers/v9/vandermaaten08a.html.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Nicolas Vecoven, Damien Ernst, and Guillaume Drion. A bio-inspired bistable recurrent cell allows for long-lasting memory. *CoRR*, abs/2006.05252, 2020. URL https://arxiv.org/abs/2006.05252.

Paul A. Viola and Michael J. Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR (1)*, pages 511–518. IEEE Computer Society, 2001. ISBN 0-7695-1272-0. URL http://dblp.uni-trier.de/db/conf/cvpr/cvpr2001-1.html#ViolaJ01.

Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Scaled-yolov4: Scaling cross stage partial network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13029–13038, June 2021.

S Wang, H Lu, P A. Dmitriev, and Z Deng. Fast object detection in compressed video. *CoRR*, abs/1811.11057, 2018. URL http://arxiv.org/abs/1811.11057.

Z Wang, X Liu, J Feng, J Yang, and H Xi. Compressed-domain highway vehicle counting by spatial and temporal regression. *IEEE Transactions on Circuits and Systems for Video Technology*, PP:1–1, 10 2017. doi: 10.1109/TCSVT.2017.2761992.

Z. Wang, X. Liu, J. Feng, J. Yang, and H. Xi. Compressed-domain highway vehicle counting by spatial and temporal regression. *IEEE Transactions on Circuits and Systems for Video Technology*, 29(1):263–274, 2019. doi: 10.1109/TCSVT.2017.2761992.

J G Wardrop. Road paper. some theoretical aspects of road traffic research. *Proceedings of the Institution of Civil Engineers*, 1(3):325–362, 1952. doi: 10.1680/ipeds.1952.11259. URL https://doi.org/10.1680/ipeds.1952.11259.

Longyin Wen, Dawei Du, Zhaowei Cai, Zhen Lei, Ming-Ching Chang, Honggang Qi, Jongwoo Lim, Ming-Hsuan Yang, and Siwei Lyu. UA-DETRAC: A new benchmark and protocol for multi-object detection and tracking. *Computer Vision and Image Understanding*, 2020.

Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3645–3649, 2017. doi: 10.1109/ICIP.2017.8296962.

A. Wong, M. Shafiee, Francis Li, and Brendan Chwyl. Tiny ssd: A tiny single-shot detection deep convolutional neural network for real-time embedded object detection. *2018 15th Conference on Computer and Robot Vision (CRV)*, pages 95–101, 2018.

C-Y Wu, M Zaheer, H Hu, R. Manmatha, A J. Smola, and P Krähenbühl. Compressed video action recognition. *CoRR*, abs/1712.00636, 2017. URL http://arxiv.org/abs/1712.00636.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016. URL http://arxiv.org/abs/1609.08144.

Kai Xu, Minghai Qin, Fei Sun, Yuhao Wang, Yen-Kuang Chen, and Fengbo Ren. Learning in the frequency domain. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Ze Yang, Shaohui Liu, Han Hu, Liwei Wang, and Stephen Lin. Reppoints: Point set representation for object detection. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9656–9665, 2019. doi: 10.1109/ICCV.2019.00975.

Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving dataset for heterogeneous multitask learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020a.

Lijun Yu, Qianyu Feng, Yijun Qian, Wenhe Liu, and Alexander G. Hauptmann. Zero-virus: Zero-shot vehicle route understanding system for intelligent transportation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020b.

Xiao Dong Yu, Ling-Yu Duan, and Qi Tian. Highway traffic information extraction from skycam mpeg video. In *Proceedings. The IEEE 5th International Conference on Intelligent Transportation Systems*, pages 37–42, 2002a. doi: 10.1109/ITSC.2002.1041185.

XiaoDong Yu, Ling-Yu Duan, and Qi Tian. Highway traffic information extraction from skycam mpeg video. In *Proceedings. The IEEE 5th International Conference on Intelligent Transportation Systems*, pages 37–42, 2002b. doi: 10.1109/ITSC.2002.1041185.

Xiaodong Yu, P. Xue, Ling yu Duan, and Q. Tian. An algorithm to estimate mean vehicle speed from mpeg skycam video. *Multimedia Tools and Applications*, 34:85–105, 2006.

Shanghang Zhang, Guanhang Wu, João Paulo Costeira, and José M. F. Moura. Understanding traffic density from large-scale web camera data. *CoRR*, abs/1703.05868, 2017. URL http://arxiv.org/abs/1703.05868.

Shifeng Zhang, Longyin Wen, Xiao Bian, Zhen Lei, and Stan Z. Li. Single-shot refinement neural network for object detection. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4203–4212, 2018. doi: 10.1109/CVPR.2018.00442.

Zhi Zhang, Tong He, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of freebies for training object detection neural networks. *CoRR*, abs/1902.04103, 2019. URL http://arxiv.org/abs/1902.04103.

Xingyi Zhou, Jiacheng Zhuo, and Philipp Krähenbühl. Bottom-up object detection by grouping extreme and center points. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 850–859, 2019.

Ying Zhou, Yu Lei, Shenghui Yang, Tao Shao, Dayong Tian, and Jiao Shi. A traffic flow estimation method based on unsupervised change detection. *Multimedia Systems*, pages 1–9, 2021.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017a.

Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable DETR: deformable transformers for end-to-end object detection. *CoRR*, abs/2010.04159, 2020. URL https://arxiv.org/abs/2010.04159.

Yousong Zhu, Chaoyang Zhao, Jinqiao Wang, X. Zhao, Yi Wu, and H. Lu. Couplenet: Coupling global structure with local parts for object detection. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 4146–4154, 2017b.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. 2017. URL https://arxiv.org/abs/1611.01578.

Zhengxia Zou, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years: A survey. *CoRR*, abs/1905.05055, 2019a. URL http://arxiv.org/abs/1905.05055.

Zhengxia Zou, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years: A survey. *arXiv preprint arXiv:1905.05055*, 2019b.