Weight parameterizations in deep neural networks

Sergey Zagoruyko

École Doctorale Paris-Est
Mathématiques & Sciences et Technologies
de l'Information et de la Communication

Thèse de doctorat
de l'Université Paris-Est
Domaine : Traitement du Signal et des Images

Présentée par

Sergey ZAGORUYKO

pour obtenir le grade de

Docteur de l'Université Paris-Est

Weight parameterizations in deep neural networks

Soutenue publiquement le 7 septembre 2018 devant le jury composé de :

Nikos KOMODAKIS	École des Ponts ParisTech	Directeur de thèse
Iasonas KOKKINOS	University College London	Rapporteur
Victor LEMPITSKY	Skolkovo Institute of Science and Technology	Rapporteur
Lourdes AGAPITO	University College London	Examinateur
Ivan LAPTEV	Inria Paris	Examinateur
Nikos PARAGIOS	École Centrale Paris	Examinateur
Renaud MARLET	École des Ponts ParisTech	Examinateur

École des Ponts ParisTech
LIGM-IMAGINE
6, av Blaise Pascal - Cité Descartes
Champs-sur-Marne
77455 Marne-la-Vallée cedex 2
France

Université Paris-Est Marne-la-Vallée
École Doctorale Paris-Est MSTIC
Département Études Doctorales
6, av Blaise Pascal - Cité Descartes
Champs-sur-Marne
77455 Marne-la-Vallée cedex 2
France

Abstract

Multilayer neural networks were first proposed more than three decades ago, and various architectures and parameterizations were explored since. Recently, graphics processing units enabled very efficient neural network training, and allowed training much larger networks on larger datasets, dramatically improving performance on various supervised learning tasks. However, the generalization is still far from human level, and it is difficult to understand on what the decisions made are based. To improve on generalization and understanding we revisit the problems of weight parameterizations in deep neural networks. We identify the most important, to our mind, problems in modern architectures: network depth, parameter efficiency, and learning multiple tasks at the same time, and try to address them in this thesis. We start with one of the core problems of computer vision, patch matching, and propose to use convolutional neural networks of various architectures to solve it, instead of manual hand-crafting descriptors. Then, we address the task of object detection, where a network should simultaneously learn to both predict class of the object and the location. In both tasks we find that the number of parameters in the network is the major factor determining it's performance, and explore this phenomena in residual networks. Our findings show that their original motivation, training deeper networks for better representations, does not fully hold, and wider networks with less layers can be as effective as deeper with the same number of parameters. Overall, we present an extensive study on architectures and weight parameterizations, and ways of transferring knowledge between them.

Keywords: neural networks, deep learning, computer vision

Résumé

Les réseaux de neurones profond ont été créés il y a plus de trois décennies avec des architectures basiques, concernant particulièrement de petits signaux ; depuis, une grande diversité de modèles et d'hyper-paramètres ont été explorés. Récemment, les calculs sur cartes graphiques ont permis aux réseaux de neurones profonds d'être appliqués à des signaux beaucoup plus grands à plus de données et ce avec des modèles significativement plus grands, cela a considérablement amélioré le rendement dans de nombreuses tâches d'apprentissage supervisé. Cependant, il existe encore une différence de performance significative avec celles obtenues par annotations humaines, cette dernière étant difficile à interpréter car elle repose sur des critères difficiles à expliciter. Pour améliorer la généralisation et la compréhension, nous ré-examinons les problèmes de paramétrage des poids des réseaux neuronaux profonds. Nous identifions et traitons les problèmes les plus importants dans les architectures modernes : la profondeur du réseau, l'efficacité des paramètres et l'apprentissage multi-tâches,. Nous commençons par l'un des problèmes fondamentaux de la vision par ordinateur, la correspondance de patch, et proposons d'utiliser des réseaux neuronaux convolutifs de différentes architectures pour le résoudre, au lieu de descripteurs prédéfinis. Ensuite, nous abordons la tâche de détection d'objets, où un réseau doit apprendre simultanément à prédire à la fois la classe de l'objet et son emplacement. Dans les deux tâches, nous constatons que le nombre de paramètres dans le réseau est le principal facteur déterminant sa performance, et nous explorons ce phénomène dans les réseaux résiduels. Nos résultats montrent qu'il existe un compromis entre profondeur et largeur des réseaux: contrairement à la pensée commune, les meilleures performances ne sont pas obtenues par les réseaux les plus profonds, par contre les performances semblent être guidées par le nombre de paramètres appris. De manière générale, nous présentons une étude empirique approfondie sur les architectures et leurs paramétrisations, ainsi que sur les moyens d'opérer un transfert d'apprentissage.

Keywords: réseau de neurones artificiels, apprentissage profond, vision par ordinateur

Acknowledgements

I'd like to thank many people who helped me along my path to writing this thesis.

I'd especially like to thank my advisor, Nikos Komodakis, for immense help and support during the work on this thesis. I am grateful to Soumith Chintala and Adam Paszke for the help with engineering in Lua Torch-7 and PyTorch frameworks. My internship at Facebook AI Research with Soumith, Adam Lerer, Sam Gross, Tsung-Yi Lin, Pedro Pinheiro and Piotr Dollár was of incredible value and source of inspiration for me. Discussions with my friends at IMAGINE were of immense help during the writing, I'd like to thank Francisco Massa, Spyros Gidaris, Martin Simonovsky, Maria Vakalopoulou, Shell Hu, Praveer Singh, Mateusz Koziński, Laura F. Julia, Marina Vinyes and Raghudeep Gadde for many fun days spent in the lab. I would also like to thank Alexander Khanin and Eugenio Culurciello for giving me temporary access to the clusters of VisionLabs and University of Purdue, without which many of the experiments would not be possible. I am grateful to everyone I collaborated with: Ivan Laptev, Josef Sivic, Alexey Dosovitskiy, Vladlen Koltun, Eugene Belilovsky, Edouard Oyallon, and others.

Finally, I'd like to thank my parents Tatiana and Nikolay Zagoruyko, who got me into engineering, and later to neural networks, and my wife Lada, for staying by my side during the difficult times of thesis writing.

Contents

List of Figures

List of Tables

Abbreviations

AP	**A**verage **P**recision
AT	**A**ttention **T**ransfer
CNN	**C**onvolutional **N**eural **N**etwork
FFT	**F**ast **F**ourier **T**ransform
GPU	**G**raphics **P**rocessing **U**nit
MLP	**M**ulti-**L**ayer **P**erceptrion
NCC	**N**ormalized **C**ross **C**orrelation
ROC	**R**eceiver **O**perating **C**haracteristic
SGD	**S**tochastic **G**radient **D**escent
SPP	**S**patial **P**yramid **P**ooling
WRN	**W**ide **R**esidual **N**etwork

Notation

This section provides a concise reference describing notation used throughout this document, taken from Goodfellow *et al.* (2016).

Numbers and Arrays

a	A scalar (integer or real)
$\boldsymbol{a}$	A vector
$\boldsymbol{A}$	A matrix
A	A tensor
$\boldsymbol{I}_n$	Identity matrix with n rows and n columns
$\boldsymbol{I}$	Identity matrix with dimensionality implied by context
$\boldsymbol{e}^{(i)}$	Standard basis vector $[0, \dots, 0, 1, 0, \dots, 0]$ with a 1 at position i
$\text{diag}(\boldsymbol{a})$	A square, diagonal matrix with diagonal entries given by $\boldsymbol{a}$
a	A scalar random variable
$\mathbf{a}$	A vector-valued random variable
$\mathbf{A}$	A matrix-valued random variable

Sets and Graphs

$\mathbb{A}$	A set
$\mathbb{R}$	The set of real numbers
$\{0, 1\}$	The set containing 0 and 1
$\{0, 1, \dots, n\}$	The set of all integers between 0 and n
$[a, b]$	The real interval including a and b
$(a, b]$	The real interval excluding a but including b
$\mathbb{A} \backslash \mathbb{B}$	Set subtraction, i.e., the set containing the elements of $\mathbb{A}$ that are not in $\mathbb{B}$
$\mathcal{G}$	A graph
$Pa_{\mathcal{G}}(\mathrm{x}_i)$	The parents of x_i in $\mathcal{G}$

Indexing

a_i	Element i of vector $\boldsymbol{a}$, with indexing starting at 1
a_{-i}	All elements of vector $\boldsymbol{a}$ except for element i
$A_{i,j}$	Element i, j of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{i,:}$	Row i of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{:,i}$	Column i of matrix $\boldsymbol{A}$
$\mathsf{A}_{i,j,k}$	Element (i, j, k) of a 3-D tensor A
$\mathsf{A}_{:,:,i}$	2-D slice of a 3-D tensor
a_i	Element i of the random vector $\mathbf{a}$

Linear Algebra Operations

$\boldsymbol{A}^\top$	Transpose of matrix $\boldsymbol{A}$
$\boldsymbol{A}^+$	Moore-Penrose pseudoinverse of $\boldsymbol{A}$
$\boldsymbol{A} \odot \boldsymbol{B}$	Element-wise (Hadamard) product of $\boldsymbol{A}$ and $\boldsymbol{B}$
$\det(\boldsymbol{A})$	Determinant of $\boldsymbol{A}$

Calculus

$\frac{dy}{dx}$	Derivative of y with respect to x
$\frac{\partial y}{\partial x}$	Partial derivative of y with respect to x
$\nabla_{\boldsymbol{x}} y$	Gradient of y with respect to $\boldsymbol{x}$
$\nabla_{\boldsymbol{X}} y$	Matrix derivatives of y with respect to $\boldsymbol{X}$
$\nabla_{\mathsf{X}} y$	Tensor containing derivatives of y with respect to X
$\frac{\partial f}{\partial \boldsymbol{x}}$	Jacobian matrix $\boldsymbol{J} \in \mathbb{R}^{m \times n}$ of $f : \mathbb{R}^n \to \mathbb{R}^m$
$\nabla^2_{\boldsymbol{x}} f(\boldsymbol{x})$ or $\boldsymbol{H}(f)(\boldsymbol{x})$	The Hessian matrix of f at input point $\boldsymbol{x}$
$\int f(\boldsymbol{x}) d\boldsymbol{x}$	Definite integral over the entire domain of $\boldsymbol{x}$
$\int_{\mathbb{S}} f(\boldsymbol{x}) d\boldsymbol{x}$	Definite integral with respect to $\boldsymbol{x}$ over the set $\mathbb{S}$

Probability and Information Theory

$\mathrm{a} \perp \mathrm{b}$	The random variables a and b are independent
$\mathrm{a} \perp \mathrm{b} \mid \mathrm{c}$	They are conditionally independent given c
$P(\mathrm{a})$	A probability distribution over a discrete variable
$p(\mathrm{a})$	A probability distribution over a continuous variable, or over a variable whose type has not been specified
$\mathrm{a} \sim P$	Random variable a has distribution P
$\mathbb{E}_{\mathrm{x} \sim P}[f(x)]$ or $\mathbb{E}f(x)$	Expectation of $f(x)$ with respect to $P(\mathrm{x})$
$\mathrm{Var}(f(x))$	Variance of $f(x)$ under $P(\mathrm{x})$
$\mathrm{Cov}(f(x), g(x))$	Covariance of $f(x)$ and $g(x)$ under $P(\mathrm{x})$
$H(\mathrm{x})$	Shannon entropy of the random variable x
$D_{\mathrm{KL}}(P \Vert Q)$	Kullback-Leibler divergence of P and Q
$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$	Gaussian distribution over $\boldsymbol{x}$ with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$

Functions

$f : \mathbb{A} \to \mathbb{B}$	The function f with domain $\mathbb{A}$ and range $\mathbb{B}$
$f \circ g$	Composition of the functions f and g
$f(\boldsymbol{x}; \boldsymbol{\theta})$	A function of $\boldsymbol{x}$ parametrized by $\boldsymbol{\theta}$. (Sometimes we write $f(\boldsymbol{x})$ and omit the argument $\boldsymbol{\theta}$ to lighten notation)
$\log x$	Natural logarithm of x
$\sigma(x)$	Logistic sigmoid, $\dfrac{1}{1+\exp(-x)}$
$\zeta(x)$	Softplus, $\log(1+\exp(x))$
$\|\|\boldsymbol{x}\|\|_p$	L^p norm of $\boldsymbol{x}$
$\|\|\boldsymbol{x}\|\|$	L^2 norm of $\boldsymbol{x}$
x^+	Positive part of x, i.e., $\max(0, x)$
$\mathbf{1}_{\text{condition}}$	is 1 if the condition is true, 0 otherwise

Sometimes we use a function f whose argument is a scalar but apply it to a vector, matrix, or tensor: $f(\boldsymbol{x})$, $f(\boldsymbol{X})$, or $f(\mathbf{X})$. This denotes the application of f to the array element-wise. For example, if $\mathbf{C} = \sigma(\mathbf{X})$, then $C_{i,j,k} = \sigma(X_{i,j,k})$ for all valid values of i, j and k.

Datasets and Distributions

p_{data}	The data generating distribution
$\hat{p}_{\text{data}}$	The empirical distribution defined by the training set
$\mathbb{X}$	A set of training examples
$\boldsymbol{x}^{(i)}$	The i-th example (input) from a dataset
$y^{(i)}$ or $\boldsymbol{y}^{(i)}$	The target associated with $\boldsymbol{x}^{(i)}$ for supervised learning
$\boldsymbol{X}$	The $m \times n$ matrix with input example $\boldsymbol{x}^{(i)}$ in row $\boldsymbol{X}_{i,:}$

Chapter 1

Introduction

Basic theory behind training deep multilayer neural networks was well developed back in the 80-90s, but, mostly due to the lack of suitable computing machines, did not see as much progress until until recently, when the good fit of graphics processing units (GPUs) to parallel nature of calculations in neural networks was noticed. Since then, the number of research works and applications of neural networks in various fields exploded, and formed a new direction of training deep networks on large amounts of data, called "deep learning".

If we look at modern deep neural networks, and compare to the ones trained in their early days, we find that there are not so many differences. Variants of stochastic gradients descent with momentum are still used for training, L^1 or L^2 are still the most effective regularization techniques, cross entropy losses are the most popular choice for training classifiers. The major change was weight reparameterization, which actually allowed successful training of deep networks.

Even five years ago training deep neural networks was very difficult for several reasons. First, it was very difficult to initialize networks such that either activations or gradients would not explode or vanish after a 5-6 layers. Second, large networks would suffer from overfitting, so strong regularization was needed. Finally, training deeper network is against the parallel nature of neural networks. However, deeper networks have the potential to build more powerful representations, useful for various tasks.

In this thesis we identify several issues with parameterizations and architectures of deep neural networks, and propose several ways to improve their efficiency and understanding. For experimental evaluation we choose computer vision tasks.

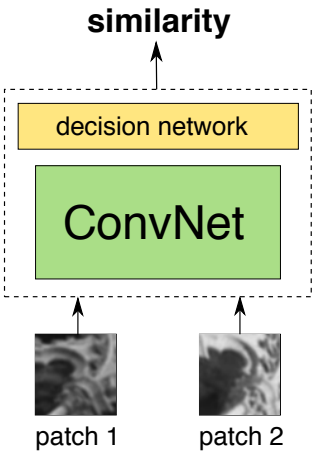


Figure 1.1: Patch matching with a convolutional neural network

We start with one of the core problems of computer vision, patch matching, which is probably one of the most fundamental tasks in computer vision and image analysis, that has given rise to the development of many hand-designed feature descriptors over the past years, including SIFT, that had a huge impact in the computer vision community. Yet, such manually designed descriptors may be unable to take into account in an optimal manner all the different factors that can affect the final appearance of image patches. On the other hand, nowadays one can easily gain access to (or even generate using available software) large datasets that contain patch correspondences between images. This begs the following question: *can we make proper use of such datasets to automatically learn a similarity function for image patches* ? Our goal is to affirmatively address the above question. We show how to learn directly from image data (i.e., without resorting to manually-designed features) a general similarity function for comparing image patches. To encode such a function, we opt for a convolutional neural network-based model that is trained to account for a wide variety of changes in image appearance. To that end, we explore and study multiple neural network architectures, including novel NCC-networks, which are specifically adapted to this task. We show that such an approach can significantly outperform the state-of-the-art on several problems and benchmark datasets. The contributions of this part are the following:

- We learn directly from image data (i.e., without any manually-designed features) a general similarity function for patches that can implicitly take into account various types of transformations and effects (due to e.g., a wide baseline, illumination, etc.).
- We explore and propose a variety of different neural network models adapted for representing such a function, highlighting at the same time network architectures that offer improved performance.

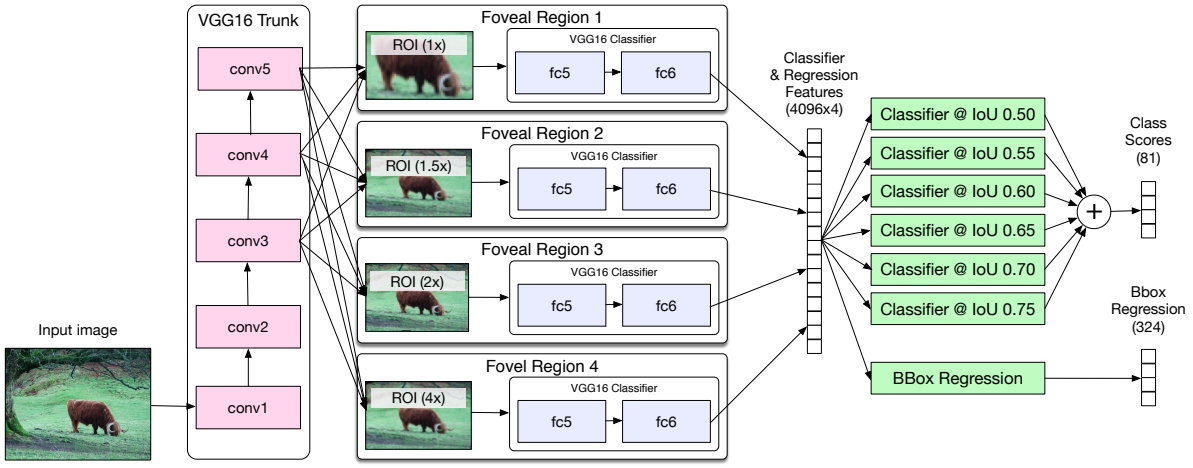
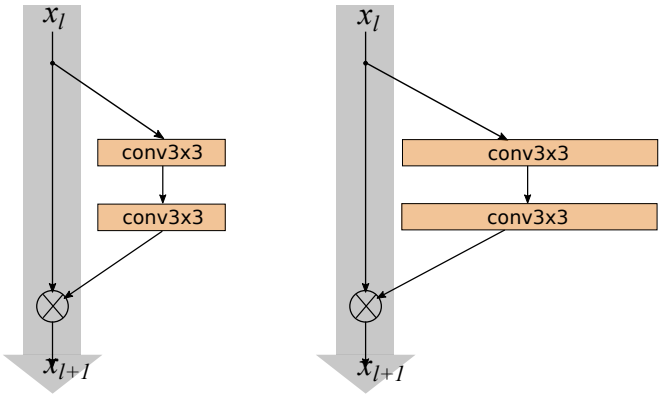


Figure 1.2: MultiPathNet model architecture

- We apply our approach on several problems and benchmark datasets, showing that it significantly outperforms the state-of-the-art and that it leads to feature descriptors with much better performance than manually designed descriptors (e.g., SIFT, DAISY) or other learnt descriptors as in Simonyan *et al.* (2014). Importantly, due to their convolutional nature, the resulting descriptors are very efficient to compute even in a dense manner.

- Last, we present NCC-networks, which are neural networks where the convolution operation is being replaced by that of normalized cross correlation, and show their significant improvements over convolutional networks in patch comparison task. We furthermore show their generality and their promising performance by presenting experimental results on the ImageNet classification task.

Then, we address the task of object detection, where a network should simultaneously learn to both predict class of the object and the location. Recognition requires network architecture to have invariance to certain transformations, reducing localization capabilities, so we propose to augment the network with multiple information flows. For experiments we choose COCO object detection dataset. Proposed in 2015, this dataset presented several new challenges for object detection over older very popular VOC datasets. In particular, it contains objects at a broad range of scales, less prototypical images, and requires more precise localization. To address these challenges, we test three modifications to the standard Fast R-CNN object detector: (1) skip connections that give the detector access to features at multiple network layers, (2) a foveal structure to exploit object context at multiple object resolutions, and (3) an integral loss function and corresponding network adjustment that improve localization. The result of these modifications is that information can flow along multiple paths in our network, including through



(a) basic (b) basic-wide

Figure 1.3: Residual and wide residual blocks

features from multiple network layers and from multiple object views. We refer to our modified classifier as a 'MultiPath' network. We couple our MultiPath network with DeepMask object proposals, which are well suited for localization and small objects, and adapt our pipeline to predict segmentation masks in addition to bounding boxes. The combined system improves results over the baseline Fast R-CNN detector with Selective Search by 66% overall and by $4\times$ on small objects. It placed second in both the COCO 2015 detection and segmentation challenges.

In both tasks of patch matching and object detection we find that number of parameters in the network is major factor determining it's performance, and explore this phenomena in residual networks. Residual networks (ResNet) proposed to reparameterize network such that to output of every pair of convolutional layers added it's input. ResNet were shown to be able to scale up to thousands of layers and still have improving performance, and achieved outstanding results on various tasks.

However, each fraction of a percent of improved accuracy costs nearly doubling the number of layers, which makes these networks very slow to train. To tackle these problems, we conduct a detailed experimental study on the architecture of ResNet blocks, based on which we propose a novel architecture where we decrease depth and increase width of residual networks. We call the resulting network structures wide residual networks (WRNs) and show that these are far superior over their commonly used thin and very deep counterparts. The contributions of this part are the following:

- We present a detailed experimental study of residual network architectures that thoroughly examines several important aspects of ResNet block structure.
- We propose a novel *widened* architecture for ResNet blocks that allows for residual networks with significantly improved performance.

- We propose a new way of utilizing dropout within deep residual networks so as to properly regularize them and prevent overfitting during training.
- Last, we show that our proposed ResNet architectures achieve state-of-the-art results on several datasets dramatically improving accuracy and speed of residual networks.

Based on this evidence, we conclude that the initial motivation behind ResNet - training deeper networks - does not fully hold, and the benefits come from increased capacity, rather than from depth. Based on this, we explore alternative definitions of ResNet, and propose an implicit skip-connection via weight parameterization as a sum of weight and Dirac delta function. This parameterization has a minor computational cost at training time and no cost at all at inference, as both Dirac parameterization and batch normalization can be folded into convolutional filters, so that network becomes a simple chain of convolution-ReLU pairs. The contributions of DiracNets part are the following:

- We propose generic Dirac weight parameterization, applicable to a wide range of neural network architectures;
- Our plain Dirac parameterized networks are able to train end-to-end with hundreds of layers. Furthermore, they are able to train with massive number of parameters and still generalize well without negative effects of overfitting;
- Dirac parameterization can be used in combination with explicit skip-connections like ResNet, in which case it eliminates the need of careful initialization.
- In a trained network Dirac-parameterized filters can be folded into a single vector, resulting in a simple and easily interpretable VGG-like network, a chain of convolution-ReLU pairs.

Finally, we explore the phenomena of knowledge distillation, allowing to transfer knowledge from a large teacher network to a smaller and more efficient student network. In addition to a common approach of using outputs of a neural network for this, we propose to use attention defined in intermediate layers, useful for understanding network predictions.

We choose attention, as it plays a critical role in human visual experience, and, furthermore, it has recently been demonstrated that it can also play an important role in the context of applying artificial neural networks to a variety of tasks from fields such as computer vision and NLP. We show that, by properly defining attention for convolutional neural networks, we can actually use this type of information in order to significantly improve the performance of a student CNN network by forcing it to mimic

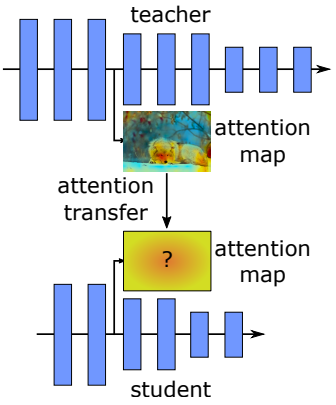


Figure 1.4: Attention transfer

the attention maps of a powerful teacher network. We propose several novel methods of transferring attention, showing consistent improvement across a variety of datasets and convolutional neural network architectures.

Contributions of this last chapter on knowledge transfer via attention maps:

- We propose attention as a mechanism of transferring knowledge from one network to another.
- We propose the use of both activation-based and gradient-based spatial attention maps.
- We show experimentally that our approach provides significant improvements across a variety of datasets and deep network architectures, including both residual and non-residual networks.
- We show that activation-based attention transfer gives better improvements than full-activation transfer, and can be combined with knowledge distillation.

1.1 Thesis outline

The document is organized as follows: Chapter 2 presents and overview of related work, Chapter 3 presents a method for learning supervised neural network for patch comparison from data, Chapter 4 presents a network for object detection in which information can follow several paths, Chapter 5 explores deep and wide residual networks for object recognition, and Chapter 6 proposes a novel way of knowledge distillation for neural networks. Finally, Chapter 7 concludes the work, presenting possible avenues for future work.

1.2 Contributions

Overall, in this thesis we present a detailed study on weight parameterizations and architectures of deep neural networks for computer vision. We propose to use convolutional neural networks for the task of patch comparison, instead of hand-crafted features, and explore various architectures and weight sharing. We also explore various architectures for the task of object detection with convolutional neural networks. In both tasks we notice some interesting properties such networks have, and focus on the understanding of depth, width, and number of parameters in residual networks. Finally, we propose a novel way of doing knowledge transfer between convolutional neural networks, using attention transfer.

All publications, software and project codes developed during this PhD are available in free access. Below are the lists of publications and corresponding codes for selected projects.

1.2.1 Publications

The work done during this PhD led to the following publications:

Peer-reviewed conferences:

- *Learning to Compare Image Patches via Convolutional Neural Networks*, Sergey Zagoruyko and Nikos Komodakis, at Computer Vision and Pattern Recognition (CVPR), 2015 Zagoruyko and Komodakis (2015);
- *A MultiPath Network for Object Detection*, Sergey Zagoruyko, Adam Lerer, Tsung-Yi Lin, Pedro O. Pinheiro, Sam Gross, Soumith Chintala and Piotr Dollár, at British Machine Vision Conference (BMVC), 2016 Zagoruyko *et al.* (2016);
- *Wide Residual Networks*, Sergey Zagoruyko and Nikos Komodakis, at British Machine Vision Conference (BMVC), 2016 Zagoruyko and Komodakis (2016b);
- *Paying More Attention to Attention: Improving the Performance of Convolutional Neural Networks via Attention Transfer*, Sergey Zagoruyko and Nikos Komodakis, at International Conference on Learning Representations (ICLR), 2017 Zagoruyko and Komodakis (2017b);
- *Scaling the Scattering Transform: Deep Hybrid Networks*, Edouard Oyallon, Eugene Belilovsky and Sergey Zagoruyko, at International Conference on Computer VisionOyallon *et al.* (2017).

- *Benchmarking Deep Learning Frameworks For The Classification Of Very High Resolution Satellite Multispectral Data*, Maria Papadomanolaki, Maria Vakalopoulou, Sergey Zagoruyko, Konstantinos Karantzalos, at ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences (ISPRS), 2016Papadomanolaki *et al.* (2016)

- *A MRF shape prior for facade parsing with occlusions*, Mateusz Kozinski, Raghudeep Gadde, Sergey Zagoruyko, Guillaume Obozinski, Renaud Marlet, at Computer Vision and Pattern Recognition (CVPR), 2015 Kozinski *et al.* (2015);

Journals:

- *Deep Compare: A Study on Using Convolutional Neural Networks to Compare Image Patches*, Sergey Zagoruyko and Nikos Komodakis, at Computer Vision and Image Understanding Special Issue: Deep Learning 2016 Zagoruyko and Komodakis (2016a).

Technical reports:

- *DiracNets: Training Very Deep Neural Networks Without Skip-Connections*, Sergey Zagoruyko and Nikos Komodakis, technical report, 2017 Zagoruyko and Komodakis (2017a).

1.2.2 Software contributions

Code for projects

- *DeepCompare*, code and pretrained models for training and applying neural networks for patch matching (Lua Torch, Caffe, Matlab, PyTorch)
  `https://github.com/szagoruyko/cvpr15deepcompare`

- *MultiPathNet*, code and pretrained models for training and applying object detection networks on COCO 2015 dataset (Lua Torch)
  `https://github.com/facebookresearch/multipathnet`

- *Wide-ResNet*, code for training wide residual networks on CIFAR, SVHN and ImageNet (Lua Torch, PyTorch)
  `https://github.com/szagoruyko/wide-residual-networks`

- *AttentionTransfer*, code for training networks with knowledge distillation and attention transfer losses on CIFAR and ImageNet (PyTorch)

  https://github.com/szagoruyko/attention-transfer

- *DiracNets*, code for training networks with Dirac parameterization (PyTorch)

  https://github.com/szagoruyko/diracnets

Chapter 2

Background

In this chapter we briefly describe operations and training methods of modern multilayer neural networks, and used in this manuscript. We mention essential operations such as convolution and activation functions, as well as more recent batch normalization and skip-connections.

2.1 Neural networks

A detailed overview of deep learning history, including supervised learning (SL), unsupervised learning (UL) with feed-forward and recurrent neural networks can be found in Schmidhuber (2015). In this chapter we include a very brief summarization of history of feed-forward networks, including convolutional, and backpropagation.

First ideas related to neural networks started to appear as early as 1800s, as first variants of linear regression methods Legendre (1805); Gauss (1809) were essentially supervised neural networks. Architectures actually referred to as neural networks, however, first appeared in 1940s McCulloch and Pitts (1943), and did not learn. SL networks, such as perceptron Rosenblatt (1958), and UL methods as self-organizing maps and associative memory Kohonen (1972); Hopfield (1982); Kohonen (1988) appeared in the following decades.

Jürgen Schmidhuber names Group Method of Data Handling (GMDH) Ivakhnenko and Lapa (1965); Ivakhnenko *et al.* (1967); Ivakhnenko (1968, 1971) one of the first methods of training deep neural networks. It had Kolmogorov-Gabor activation functions, could be trained with 8 layers and used now traditional data split. He also names later Neocognitron Fukushima (1980) the first deep artificial neural network and the first to incorporate the neurophysiological insights. It was also the first convolutional neural network, on which we continue in section 2.1.2.

We briefly describe multilayer neural networks and their building blocks in the following subchapters.

2.1.1 Multi-layer perceptron

Multi-layer neural network is defined by a vector of parameters $\boldsymbol{\theta}$ and a function f of inputs $\boldsymbol{x}$ and $\boldsymbol{\theta}$. The function and parameters are typically split into simpler operations, called layers. Normally, $f(\boldsymbol{x}, \boldsymbol{\theta})$ is trained to approximate some function $g(\boldsymbol{x})$, with a loss $\mathcal{L}$ defined on outputs of $f(\boldsymbol{x}, \boldsymbol{\theta})$, optimizing which involves doing gradient descent using gradients of $\mathcal{L}$ w.r.t. $\boldsymbol{\theta}$ computed via chain rule, or *backpropagation*, which we describe in section 2.2.

Let's review a basic neural network, *multi-layer perceptron (MLP)* with a single hidden layer. Let it have vector of parameters $\boldsymbol{\theta} = \text{vec}(\boldsymbol{W}, \boldsymbol{v}, \boldsymbol{b})$, where $\boldsymbol{W}$ is a 2-dimensional weight matrix of the hidden layer with biases vector $\boldsymbol{b}$, $\boldsymbol{v}$ weight vector of the output layer, and vec is a vectorization function. Let it also have sigmoid activation function $\sigma(\boldsymbol{h})$ of outputs of previous layer $\boldsymbol{h}$. Function of input vector $\boldsymbol{x}$ and $\boldsymbol{\theta}$

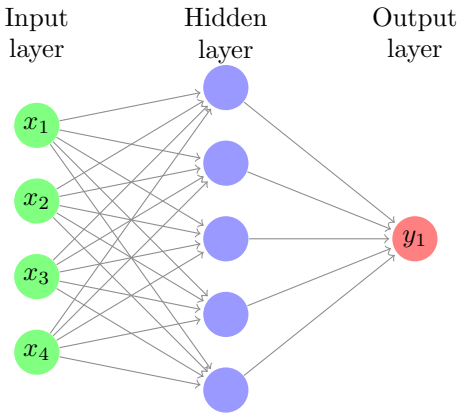


Figure 2.1: MLP with a single hidden layer.

defining MLP is the following:

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sum_{i=1}^{N} v_i \sigma(\boldsymbol{w}_i^\top \boldsymbol{x} + b_i).$$

Schematic representation of such function can be found on figure 2.1. Theoretically, such function is able to approximate any function Cybenko (1989); Hornik *et al.* (1989), given enough (possibly very large number) of neurons in the hidden layer $\boldsymbol{W}$. It does not define if such network is learnable, though. In practice, neural networks are defined by a more complex combinations of layers, as training MLP on highly multidimensional data is often too costly or infeasible.

2.1.2 Convolutional neural networks

The first predecessor of modern convolutional neural networks was Neocognitron Fukushima (1980), heavily inspired from mammalian visual cortex. In a simplified form, multilayer convolutional neural networks for document recognition were proposed in LeCun *et al.* (1998). Convolutional neural networks introduce weight sharing to the matrix multiplication as in MLP, and allow efficient approximation by doing operations in local neighborhoods of data and activations with shared parameters. Despite being introduced almost two decades ago, basic architecture and means of training remain almost unchanged even today. For example, very similar convolutional neural network was used by Krizhevsky *et al.* (2012b) to win ImageNet 2012 competition with their famous AlexNet architecture. We further review the most common components of modern convolutional neural networks: convolutional layer, activation function, and batch normalization.

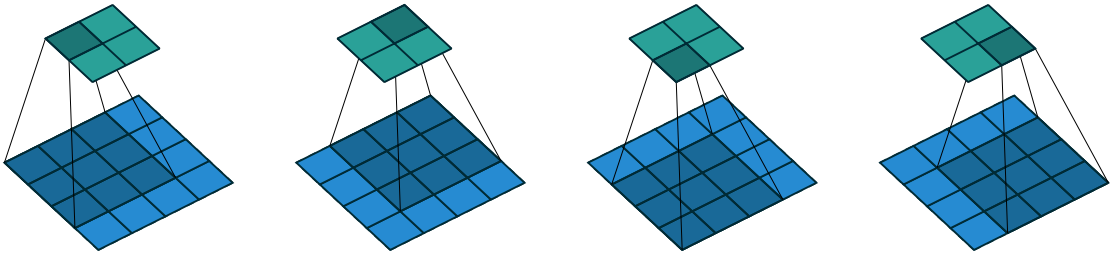


Figure 2.2: Schematic representation of convolving input of size 4×4 (blue) with filters of size 3×3, output is 2×2 (cyan) Dumoulin and Visin (2016)

Let $\boldsymbol{I}$ be output of previous hidden layer or input image (input to current layer), $\boldsymbol{S}$ output of the current layer, and $\boldsymbol{K}$ be the filters of the current layer. We define convolution operation by $*$ symbol:

$$\boldsymbol{S} = \boldsymbol{K} * \boldsymbol{I}$$

For a single plane 2-dimensional input $\boldsymbol{I}$ and filter $\boldsymbol{K}$ of size $N \times M$ with no padding output at position k, m is a discrete cross correlation of $\boldsymbol{I}$ and $\boldsymbol{K}$:

$$\boldsymbol{S}(k, m) = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} \boldsymbol{K}(i, j)\boldsymbol{I}(k + i, m + j)$$

See also visualization on fig. 2.2. Various modifications for neural networks exist, such as depthwise, grouped, dilated, etc., we refer reader to Dumoulin and Visin (2016) for more details and explanations. For backpropagation derivatives w.r.t. $\boldsymbol{I}$ and $\boldsymbol{K}$ need to be computed, which are convolutions themselves.

2.1.3 Activation function

Activation is an essential operation adding complexity and capacity to the network. It needs to be differentiable and nonlinear (not necessarily continuously differentiable) to work with backpropagation. It is typically an elementwise function of output of the previous layer $\boldsymbol{x}$, e.g. sigmoid activation function:

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

Recently, ReLU gained popularity over hyperbolic tangent and sigmoid functions, due to simplicity and improved convergence and generalization. It is a simple thresholding operation (fig. 2.3):

$$g(x) = \max(0, x)$$

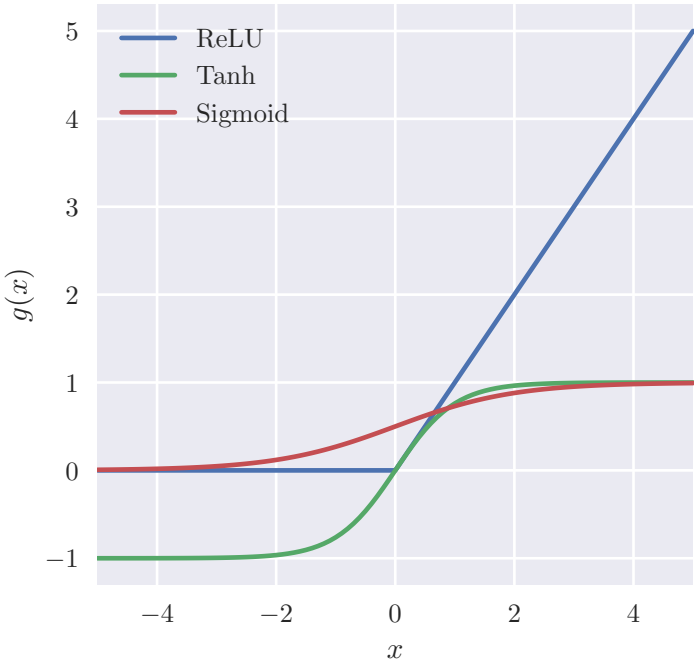


Figure 2.3: ReLU, Sigmoid and Tanh activation functions.

It backpropagates error $\boldsymbol{e}$ where $x > 0$:

$$\frac{\partial g}{\partial x} e = \begin{cases} e, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$$

so that both output and gradient have a lot of zeros, i.e. their distribution is significantly different from the distribution of input.

Other nonlinearities were proposed, such as ELU Clevert *et al.* (2015), parametric ReLU He *et al.* (2015) and leaky ReLU Maas *et al.* (2013), and search continues, but in practice changing nonlinearity brings no or marginal benefits over ReLU.

Let's analyze a network which is a sequential chain of convolution-ReLU layers (fig. 2.4):

$$\boldsymbol{y} = g(\boldsymbol{W}_n * g(\boldsymbol{W}_{n-1} * g(...\boldsymbol{W}_1 * \boldsymbol{x}...)))$$

If we would assume $\boldsymbol{W}$ and input to the network are drawn from normal distribution $\mathcal{N}(0, 1)$, activations would quickly diminish to zeroes. Even with weight initialization that aims to preserve activation or error variance through the network such as He *et al.* (2015), it is difficult to preserve both. Batch normalization in the following section significantly simplifies initialization.

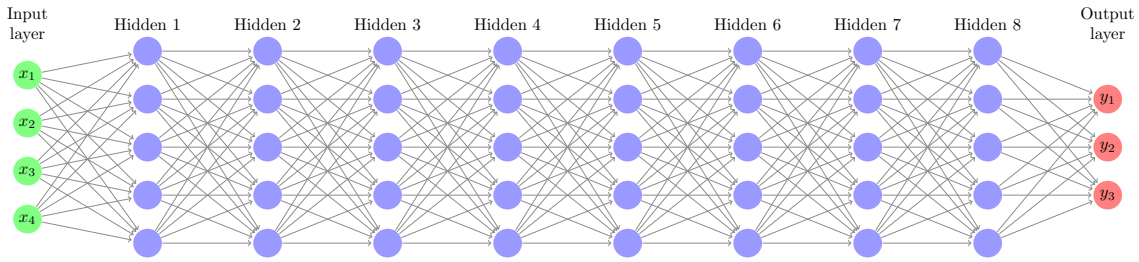


Figure 2.4: Deep neural network with 8 hidden layers.

2.1.4 Batch normalization

Batch normalization Ioffe and Szegedy (2015) aims to remove internal covariate shift by performing batch-wise mean and std normalization. Statistics are computed over minibatch dimension. Let $\boldsymbol{x}$ be a minibatch of size m, it's per-output mean and variance:

$$\mu = \frac{1}{m}\sum_{i}^{m} x_i$$
$$\sigma^2 = \frac{1}{m}\sum_{i}^{m}(x_i - \mu)^2$$

Batch normalization then performs:

$$y_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}\gamma + \beta,$$

where γ and β are per-output scaling and bias coefficients.

It has been shown that batch normalization speed up convergence and significantly improves generalization, increasing network capacity at the same time. Also, it allows setting large learning rates without worrying about divergence. However, it complicates network structure, makes multi-GPU training more difficult, and has different formulation in training/validation phases, so several alternatives were proposed, such as weight normalization Salimans and Kingma (2016) and layer normalization Ba *et al.* (2016), which, however, do not work as well in practice.

2.1.5 Skip-connections

Skip connection in it's simplest form is a reparameterization which does addition of the layer output to input, instead of simply propagating it further, e.g. hidden layer with sigmoid nonlinearity and skip-connection:

$$\boldsymbol{y} = \boldsymbol{x} + \sigma(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$

It can also be implemented in concatenation of input signal and output, or as a gated summation, for example, as in LSTM Hochreiter and Schmidhuber (1997), or in Highway networks Srivastava *et al.* (2015). ResNet basic building block actually has two linear-activation pairs in the residual part, and batch normalization is essential for it to work.

2.2 Neural network training methods

Fitting deep neural networks with aforementioned blocks is typically done by optimizing a non-convex objective function, a procedure called *training*. Let us briefly review the problem and the common methods used to solve it.

Let $\{\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\}, i = 1..N$ be a set $\mathbb{X}$ of N training pairs of inputs and labels, and $\boldsymbol{\theta}$ the parameters of the neural network defined as a function $f(\boldsymbol{x}; \boldsymbol{\theta})$. The learning problem is then fitting $\boldsymbol{\theta}$ into training data:

$$\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) + \lambda \|\boldsymbol{\theta}\|_p, \tag{2.1}$$

where $\mathcal{L}(\hat{\boldsymbol{y}}^{(i)}, \boldsymbol{y}^{(i)})$ is a loss function of predicted label $\hat{\boldsymbol{y}}^{(i)}$ and target label $\boldsymbol{y}^{(i)}$. Second term adds L^p-regularization with coefficient λ. Most commonly used is L^2-regularization, also referred to as weight decay.

If $f(\boldsymbol{x}; \boldsymbol{\theta})$ is defined by a multilayer neural network, 2.1 can be efficiently optimized using gradient descent and backpropagation. Gradient descent in parameter space in context of Euler-LaGrange equations was discussed since the 1960s Bryson (1961); Kelley (1960); Pontryagin *et al.* (1961), and efficient error backpropagation in arbitrary, discrete NN-like networks was proposed in Linnainmaa (1970, 1976), and was used to minimize control parameters in Dreyfus (1973). According to Schmidhuber (2015), the first NN-specific application of backpropagation was described in Werbos (1981), and later in Parker (1985); Lecun (1987); LeCun (1988).

Despite the existence of efficient second order methods Becker and LeCun (1989); Martens and Grosse (2015), simple minibatch stochastic gradient descent (SGD) with momentum Polyak (1964) remains the most efficient and commonly used optimization methods for deep neural networks. Simple minibatch SGD is done by updates:

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \frac{\eta_k}{n} \nabla f_i(\boldsymbol{\theta}_k), \tag{2.2}$$

where η_k and n are learning rate at step k and minibatch size respectively. With the addition of the velocity vector $\boldsymbol{v}$ the update rule becomes:

$$\boldsymbol{v}_{k+1} \leftarrow \alpha_k \boldsymbol{v}_k - \frac{\eta_k}{n} \nabla f_i(\boldsymbol{\theta}_k),$$
$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k + \boldsymbol{v}_{k+1},$$

which is the update rule of SGD with momentum.

In case $f(\boldsymbol{\theta}_k)$ is strongly convex or convex and smooth, there are convergence guarantees for SGD update rules. However, neural network functions are highly non-convex, so it is unclear how to optimally set learning rate rule η_k. Often used in practice are exponentially decaying η_k at every step, of every $m > 1$ steps. Several adaptive learning rates were proposed, which can be advantageous by setting different learning rates for different coordinates, e.g. larger learning rates for coordinates with smaller gradients, and smaller for larger gradients, such as AdaGrad Duchi *et al.* (2010), RMSProp, Adam Kingma and Ba (2014) and others. Such methods result in biased gradient updates which change the underlying optimization problem Wilson *et al.* (2017), and often end up with worse generalization error compared to simple SGD.

Chapter 3

Using convolutional neural networks to compare image patches

Comparing patches across images is probably one of the most fundamental tasks in computer vision and image analysis, that has given rise to the development of many hand-designed feature descriptors over the past years, including SIFT, that had a huge impact in the computer vision community. Yet, such manually designed descriptors may be unable to take into account in an optimal manner all the different factors that can affect the final appearance of image patches. On the other hand, nowadays one can easily gain access to (or even generate using available software) large datasets that contain patch correspondences between images. This begs the following question: *can we make proper use of such datasets to automatically learn a similarity function for image patches* ? Our goal in this work is to affirmatively address the above question. We show how to learn directly from image data (i.e., without resorting to manually-designed features) a general similarity function for comparing image patches. To encode such a function, we opt for a CNN-based model that is trained to account for a wide variety of changes in image appearance. To that end, we explore and study multiple neural network architectures, including novel NCC-networks, which are specifically adapted to this task. We show that such an approach can significantly outperform the state-of-the-art on several problems and benchmark datasets.

This chapter is based on Deep Compare: A Study on Using Convolutional Neural Networks to Compare Image Patches Zagoruyko and Komodakis (2016a).

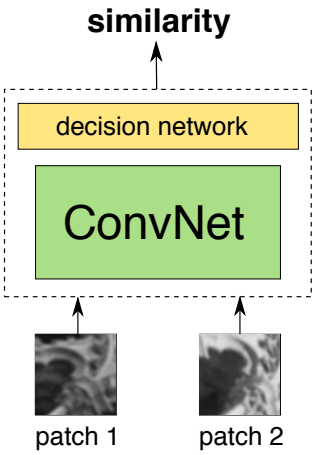


Figure 3.1: Our goal is to learn a general similarity function for image patches. To encode such a function, here we make use of and explore convolutional neural network architectures.

3.1 Introduction

Comparing patches is a subroutine that plays an important role in a wide variety of vision tasks. These can range from low-level tasks such as structure from motion, wide baseline matching, building panoramas, and image super-resolution, up to higher-level tasks such as object recognition, image retrieval, and classification of object categories, to mention a few characteristic examples. Of course, the problem of deciding if two patches correspond to each other or not is quite challenging as there exist far too many factors that affect the final appearance of an image Nowak and Jurie (2007). These can include changes in viewpoint, variations in the overall illumination of a scene, occlusions, shading, differences in camera settings, etc. Many hand-crafted feature descriptors were designed for this task, including SIFT Lowe (2004), which, however, may not be able to take into account in an optimal manner all of the aforementioned factors that determine the appearance of a patch. For this reason, in this work we aim to explore if it is possible to generate a patch similarity function *from scratch*, i.e., without attempting to use any manually designed features but instead directly learning this function from annotated pairs of raw image patches.

To that end, inspired also by the recent advances in neural architectures and deep learning, we choose to represent such a function in terms of a deep convolutional neural network LeCun (1988); Krizhevsky *et al.* (2012b) (Fig. 3.1) with the help of large data collections of patch correspondences Snavely *et al.* (2008). Given that there exist several ways in which patch pairs can be processed by the network or in which the information sharing can take place, we are also interested in addressing the issue of what specific network architectures are best to be used in a task like this. We thus explore and propose various types of networks, having architectures that exhibit different trade-offs and advantages. This includes

networks such as: (i) *siamese* (this type of network resembles the idea of having a descriptor, in which case there are two branches – one per patch – in the network that share exactly the same architecture and the same set of weights), (ii) *pseudo-siamese* (as siamese, but without sharing weights between branches, the reason for which will be explained later), (iii) *2-channel* (where, unlike previous models, there is no direct notion of descriptor in the architecture and the network proceeds directly with the similarity estimation), (iv) *central-surround two-stream* (where we modify the network to consist of two separate streams, central and surround, which enable a processing in the spatial domain that takes place over two different resolutions), (v) *spatial-pyramid-pooling (SPP)*, (vi) *deep networks*, and (vii) *NCC-networks* (these will be explained later). Many of the above variations can be used in conjunction with each other, thus leading to a wide range of models for comparing patches. Based on these, we draw interesting conclusions about which architectural choices help in improving performance in practice.

In all of the above cases, to train the proposed networks we are using as sole input a large database that contains pairs of raw image patches (both matching and non-matching). This allows one to easily further improve the performance of the proposed methodology for comparing patches simply by enriching such a database with more samples, where software for automatically generating such samples can be readily available Snavely *et al.* (2006).

This work extends Zagoruyko and Komodakis (2015) by providing a more complete study of the architecture of patch matching convolutional networks along with new experimental results. In addition, inspired by normalized cross correlation (NCC) and the fact that convolutional networks from Zagoruyko and Komodakis (2015) lack normalization compared to SIFT, we develop NCC-networks, performing normalized cross correlation instead of convolution, and learning NCC-filters. We show that NCC-networks achieve significantly better results than convolutional. To our knowledge, this is the first attempt to train such networks, and to show their generality we also present corresponding results on the ImageNet classification task.

To conclude this section, the chapter's main contributions are as follows:

- We learn directly from image data (i.e., without any manually-designed features) a general similarity function for patches that can implicitly take into account various types of transformations and effects (due to e.g., a wide baseline, illumination, etc.).
- We explore and propose a variety of different neural network models adapted for representing such a function, highlighting at the same time network architectures that offer improved performance.

- We apply our approach on several problems and benchmark datasets, showing that it significantly outperforms the state-of-the-art and that it leads to feature descriptors with much better performance than manually designed descriptors (e.g., SIFT, DAISY) or other learnt descriptors as in Simonyan *et al.* (2014). Importantly, due to their convolutional nature, the resulting descriptors are very efficient to compute even in a dense manner.

- Last, we present NCC-networks, which are neural networks where the convolution operation is being replaced by that of normalized cross correlation, and show their significant improvements over convolutional networks in patch comparison task. We furthermore show their generality and their promising performance by presenting experimental results on the ImageNet classification task.

3.2 Related work

The conventional approach to compare patches is to use descriptors and a squared euclidean distance. Most feature descriptors are hand-crafted as SIFT Lowe (2004), SURF Bay *et al.* (2006), DAISY Tola *et al.* (2008), ORB Rublee *et al.* (2011), or even created with randomization as BRISK Leutenegger *et al.* (2011) or BRIEF Calonder *et al.* (2010). Recently, methods for learning a descriptor have been proposed Trzcinski *et al.* (2012, 2013), (e.g., DAISY-like descriptors learn pooling regions and dimensionality reduction Brown *et al.* (2011)), Simonyan *et al.* Simonyan *et al.* (2014) proposed a convex procedure for training on both tasks.

Our approach, however, is inspired by the recent success of convolutional neural networks Razavian *et al.* (2014); Taigman *et al.* (2014); Szegedy *et al.* (2013); Eigen *et al.* (2014). Although these models involve a highly non-convex objective function during training, they have shown outstanding results in various tasks Razavian *et al.* (2014). Fischer *et al.* Fischer *et al.* (2014) analysed the performance of convolutional descriptors from AlexNet network (that was trained on Imagenet dataset Krizhevsky *et al.* (2012b)) on the well-known Mikolajczyk dataset Mikolajczyk and Schmid (2005) and showed that these convolutional descriptors outperform SIFT in most cases except blur. They also proposed an unsupervised training approach for deriving descriptors that outperform both SIFT and Imagenet trained network.

Zbontar and LeCun in Zbontar and LeCun (2015) have recently proposed a CNN-based approach to compare patches for computing cost in small baseline stereo problem and shown the best performance in KITTI dataset. However, the focus of that work was only on comparing pairs that consist of very small

patches like the ones in narrow baseline stereo. In contrast, here we aim for a similarity function that can account for a broader set of appearance changes and can be used in a much wider and more challenging set of applications, including, e.g., wide baseline stereo, feature matching and image retrieval.

After publication Zagoruyko and Komodakis (2015) served as baseline for a number of successful applications of convolutional neural networks in descriptor learning for keypoint matching, stereo-matching, optical flow computation and tracking Bertinetto *et al.* (2016); Tao *et al.* (2016). Among them G *et al.* (2016) and Balntas *et al.* (2016) improved results of siamese networks in patch matching by using triplet loss functions, building on Hoffer and Ailon (2015), leading to more efficient architectures and better training time. Hard negative mining and sampling strategies for training deep convolutional descriptors were explored in Simo-Serra *et al.* (2015). These improvements are complementary and can be combined with our work. Problem of assigning orientations to convolutional descriptors was explored in Yi *et al.* (2016a). Directly learning the whole pipeline of detecting and matching keypoints was explored in Yi *et al.* (2016b); Choy *et al.* (2016). Concurrently to our work, MatchNet Han *et al.* (2015) was released with the same goal as ours. Our networks outperform MatchNet having lower descriptor dimensionality, except in 4096d case, which we did not explore because of its implausible application.

Cross-correlation based descriptors were among the first descriptors used in computer vision Mikolajczyk and Schmid (2005), and are still popular in stereo matching Faugeras *et al.* (1993); Goesele *et al.* (2007), because of their simplicity and ease of computation. It also offers invariance to illumination changes, but poor invariance to affine transformations, which we propose to address by learning NCC-filters. Compared to batch normalization Ioffe and Szegedy (2015) NCC-layer normalizes inputs across a different dimension.

The remainder of the chapter is structured as follows. We first describe in section 3.3 a wide variety of architectures that can be used to build patch comparison networks (many of which are possible to be combined with each other), we then provide in section 3.4 details about the training process that was followed, and finally in section 3.5 we evaluate the proposed networks on different datasets and tasks, including patch comparing, wide baseline stereo estimation, descriptor evaluation and image classification.

3.3 Architectures

As already mentioned, the input to the neural network is considered to be a pair of image patches. Our models do not impose any limitations with respect to the number of channels in the input patches,

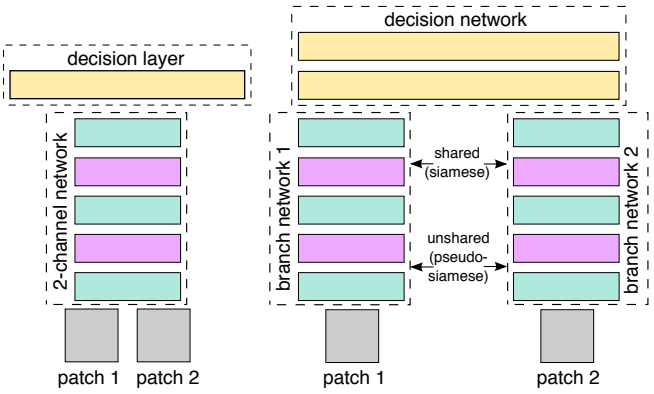


Figure 3.2: Three basic network architectures: 2-channel on the left, siamese and pseudo-siamese on the right (the difference between siamese and pseudo-siamese is that the latter does not have shared branches). Color code used: cyan = Conv+ReLU, purple = max pooling, yellow = fully connected layer (ReLU exists between fully connected layers as well).

i.e., given a dataset with colour patches the networks could be trained to further increase performance. However, to be able to compare our approach with state-of-the-art methods on existing datasets, we chose to use only grayscale patches during training. Furthermore, with the exception of the SPP model described in section 3.3.2, in all other cases the patches given as input to the network are assumed to have a fixed size of 64×64 (this means that original patches may need to be resized to the above spatial dimensions).

There are several ways in which patch pairs can be processed by the network and how the information sharing can take place in this case. For this reason, we explored and tested a variety of models. We start in section 3.3.1 by describing the three basic neural network architectures that we studied, i.e., 2-channel, Siamese, Pseudo-siamese (see Fig. 3.2), which offer different trade-offs in terms of speed and accuracy (note that, as usually, applied patch-matching techniques imply testing a patch against a big number of other patches, and so re-using computed information is always useful). Essentially these architectures stem from the different way that each of them attempts to address the following question: when composing a similarity function for comparing image patches, do we first choose to compute a descriptor for each patch and then create a similarity on top of these descriptors or do we perhaps choose to skip the part related to the descriptor computation and directly proceed with the similarity estimation?

In addition to the above basic models, we also describe in section 3.3.2 extra variations concerning the network architecture. These variations, which are not mutually exclusive to each other, can be used in conjunction with any of the basic models described in section 3.3.1. Overall, this leads to a variety of models that is possible to be used for the task of comparing image patches.

3.3.1 Basic models

Siamese: This type of network resembles the idea of having a descriptor Bromley *et al.* (1993); Chopra *et al.* (2005). There are two branches in the network that share exactly the same architecture and the same set of weights. Each branch takes as input one of the two patches and then applies a series of convolutional, ReLU and max-pooling layers. Branch outputs are concatenated and given to a top network that consists of linear fully connected and ReLU layers. In our tests we used a top network consisting of 2 linear fully connected layers (each with 512 hidden units) that are separated by a ReLU activation layer. Branches of the siamese network can be viewed as descriptor computation modules and the top network - as a similarity function. For the task of matching two sets of patches at test time, descriptors can first be computed independently using the branches and then matched with the top network (or even with a distance function like l_2).

Pseudo-siamese: In terms of complexity, this architecture can be considered as being in-between the siamese and the 2-channel networks. More specifically, it has the structure of the siamese net described above except that the weights of the two branches are uncoupled, i.e., not shared. This increases the number of parameters that can be adjusted during training and provides more flexibility than a restricted siamese network, but not as much as the 2-channel network described next. On the other hand, it maintains the efficiency of siamese network at test time.

2-channel: unlike the previous models, here there is no direct notion of descriptor in the architecture. We simply consider the two patches of an input pair as a 2-channel image, which is directly fed to the first convolutional layer of the network. In this case, the bottom part of the network consists of a series of convolutional, ReLU and max-pooling layers. The output of this part is then given as input to a top module that consists simply of a fully connected linear decision layer with 1 output. This network provides greater flexibility compared to the above models as it starts by processing the two patches jointly. Furthermore, it is fast to train, but in general at test time it is more expensive as it requires all combinations of patches to be tested against each other in a brute-force manner.

We further denote siamese networks with `siam` prefix, and 2-channel - `2ch`.

3.3.2 Additional models

In this section we provide additional architectures, that can be combined with the above, and describe the structure of these networks in more detail. In doing so, our goal is to indicate specific architectural

choices that are beneficial for the task of comparing patches. We thus explore how this task is affected by the depth of the network (section 3.3.2) or by reducing the number of parameters through the use of average pooling before the classification layer (section 3.3.2), we explore novel NCC-networks that rely on the use of normalized cross correlation operations (section 3.3.2), central-surround networks that rely on exploting information from multiple resolutions (section 3.3.2) as well as networks that properly make use of spatial pyramid pooling when comparing patches (section 3.3.2).

Deep networks

Convolutional neural networks have seen a gradual increase of the number of layers in the last few years, starting from AlexNet Krizhevsky *et al.* (2012b), VGG Simonyan and Zisserman (2015), Inception Szegedy *et al.* (2015) to Residual He *et al.* (2016a) networks, corresponding to improvements in many image recognition tasks. The superiority of deep networks has been spotted in several works in the recent years Bianchini and Scarselli (2014); Montúfar *et al.* (2014). Inspired by this, we apply the technique proposed by Simonyan and Zisserman in Simonyan and Zisserman (2015) advising to break up large convolutional layers into smaller 3x3 kernels, separated by ReLU activations, which is supposed to increase the nonlinearities inside the network and make the decision function more discriminative. They also report that it might be difficult to initialize such a network, we, however, do not observe this behavior and train the network from scratch as usual. In our case, when applying this technique to our model, the convolutional part of the final architecture turns out to consist of one convolutional 4x4 layer and 6 convolutional layers with 3x3 layers, separated by ReLU activations. As we shall also see later in the experimental results, such a change in the network architecture can contribute in further improving performance. We also tried to utilize residual connections as proposed in He *et al.* (2016a), but did not observe improvements probably due to simplicity of our task and less deep base network architecture.

2-channel-avg

This network architecture (further `2ch-avg`) is similar to 2-channel, but with average pooling on top before the final layer, which was shown to effectively reduce the number of parameters reducing the risk of overfitting Lin *et al.* (2013), while at the same time speeding up the network. We note that average pooling before the final decision layer is used in many recent successful architectures such as Inception Szegedy *et al.* (2015) and ResNet He *et al.* (2016a).

Central-surround two-stream networks

As its name suggests, the proposed architecture consists of two separate streams, central and surround, which enable a processing in the spatial domain that takes place over two different resolutions. More specifically, the central high-resolution stream receives as input two 32×32 patches that are generated by cropping (at the original resolution) the central 32×32 part of each input 64×64 patch. Furthermore, the surround low-resolution stream receives as input two 32×32 patches, which are generated by downsampling at half the original pair of input patches. The resulting two streams can then be processed by using any of the basic architectures described in section 3.3.1 (see Fig. 3.3 for an example that uses a siamese architecture for each stream).

One reason to make use of such a two-stream architecture is because multi-resolution information is known to be important in improving the performance of image matching. Furthermore, by considering the central part of a patch twice (i.e., in both the high-resolution and low-resolution streams) we implicitly put more focus on the pixels closer to the center of a patch and less focus on the pixels in the periphery, which can also help for improving the precision of matching (essentially, since pooling is applied to the downsampled image, pixels in the periphery are allowed to have more variance during matching). Note that the total input dimensionality is reduced by a factor of two in this case. As a result, training proceeds faster, which is also one other practical advantage.

NCC networks

Normalization is an important part of many hand-crafted descriptors, due to various brightness, illumination, contrast, etc. conditions of patches coming from real images. Even a simple NCC-metric can be used to efficiently compare image patches. While normalization in convolutional networks can be achieved by adding certain normalization layers to some extent, we choose a different approach, incorporating NCC into convolutional layers directly, defining convolutional neural network consisting of NCC-layers, which we call NCC-network. Each NCC-layer normalizes input data and its weights and learns correlation filter coefficients. Any of the architectures above can be combined with NCC-networks, and, as we show in experimental results, NCC-networks significantly outperform convolutional networks in patch matching. On fig. 3.4b we present `2ch-ncc-avg` schematic representation.

Below we describe the structure of NCC-layer. Normalized cross-correlation of discrete signals x and y is defined as:

$$NCC(x, y) = \sum_{i=1}^{n} \frac{x_i - \mu_x}{\sigma_x + \epsilon} \frac{y_i - \mu_y}{\sigma_y + \epsilon} \quad , \tag{3.1}$$

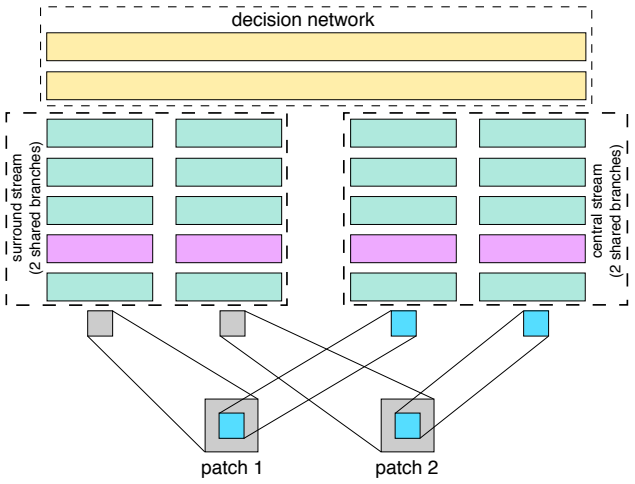


Figure 3.3: A central-surround two-stream network that uses a siamese-type architecture to process each stream. This results in 4 branches in total that are given as input to the top decision layer (the two branches in each stream are shared in this case).

where μ_x, μ_y and σ_i, σ_y are mean and std of signals x and y correspondingly

$$\mu_x = \sum_{i=1}^{n} \frac{x_i}{n}, \sigma_x = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_x)^2} \ , \tag{3.2}$$

and ϵ is a small constant added to avoid numerical issues.

In a two-dimensional case when input tensor x has D features and size HxW and NCC-layer with N neurons and filters of size KxK, then μ_x, σ_x are computed over spatial neighbourhoods of x with size KxK and feature dimension, while μ_w, σ_w of NCC-filters w are computed over DxKxK dimensions, resulting in exactly the same tensor size as in convolutional case.

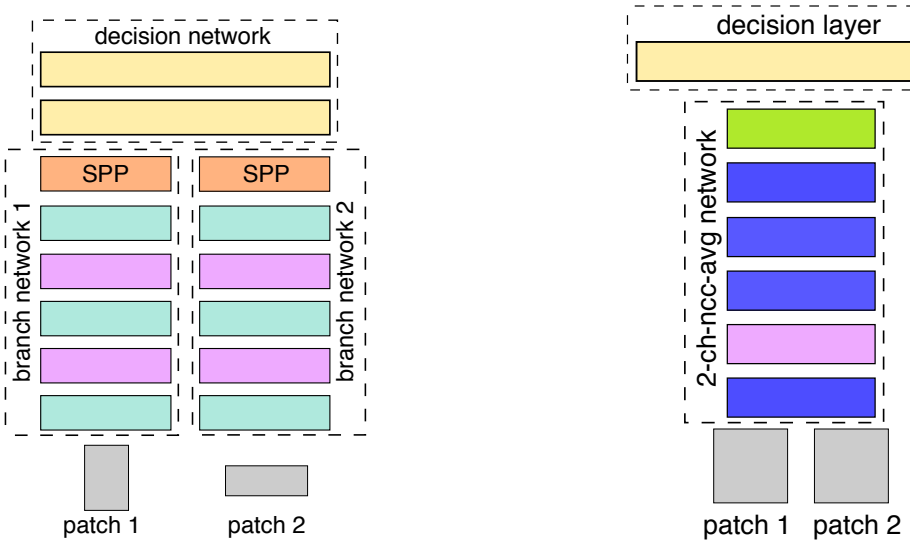
We define NCC operation on input tensor x with filters w and biases b:

$$y = NCC(x, w) + b = \sum_{i=1}^{n} \frac{x_i - \mu_x}{\sigma_x + \epsilon} \frac{w_i - \mu_w}{\sigma_w + \epsilon} + b \ . \tag{3.3}$$

The bias is added to deactivate outputs with negative correlations, which are thresholded by activation function, e.g. ReLU, if negative.

When backpropagating errors through NCC-layers gradients w.r.t. to inputs and weights chain rule is used. Let:

$$w_i^\star = \frac{w_i - \mu_w}{\sigma_w + \epsilon}, x_i^\star = \frac{x_i - \mu_x}{\sigma_x + \epsilon} \tag{3.4}$$



(a) spp (b) 2ch-ncc-avg

Figure 3.4: SPP and NCC architectures. SPP network for a siamese architecture: SPP layers (orange) are inserted immediately after the 2 branches of the network so that the top decision layer has an input of fixed dimensionality for any size of the input patches. Color codes used in NCC-scheme: blue = NCC+ReLU, green = avg pooling

Then the derivative of y w.r.t to input x_i is:

$$\frac{\delta y}{\delta x_i} = \frac{w_i^\star}{\sigma_x + \epsilon} - \frac{x_i \sum_{i=1}^{n} w_i^\star x_i}{(\sigma_x + \epsilon)^3} \tag{3.5}$$

The second part of equation 3.5 contains output y and is simplified to:

$$\frac{\delta y}{\delta x_i} = \frac{w_i^\star}{\sigma_x + \epsilon} - \frac{x_i}{(\sigma_x + \epsilon)^2} y. \tag{3.6}$$

Derivatives w.r.t. to weights are computed the same way containing output y variable in the second part, so both derivatives w.r.t. inputs and weights can be computed in an efficient manner.

Unlike convolutional, NCC-networks initialization distribution parameters have no effect on forward propagation, as weights are normalized. However it does make difference on backward propagation, so we derived and empirically adjusted initialization technique for NCC-layers, following He *et al.* (2015). We use zero-mean Gaussian distribution whose standard deviation equals to $\sqrt{10 n_l}$ where n_l is a number of feature planes in the layer. If NCC-layer has batch normalization we set standard deviation to 1.

Spatial pyramid pooling (SPP) networks

Up to this point we have been assuming that the network requires the input patches to have a fixed size of 64×64. This requirement comes from the fact that the output of the last convolutional layer of the network needs to have a predefined dimensionality. Therefore, when we need to compare patches of arbitrary sizes, this means that we first have to resize them to the above spatial dimensions. However, if we look at the example of descriptors like SIFT, for instance, we can see that another possible way to deal with patches of arbitrary sizes is via adjusting the size of the spatial pooling regions to be proportional to the size of the input patch so that we can still maintain the required fixed output dimensionality for the last convolutional layer without deteriorating the resolution of the input patches.

This is also the idea behind the recently proposed SPP-net architecture He *et al.* (2014), which essentially amounts to inserting a spatial pyramid pooling layer between the convolutional layers and the fully-connected layers of the network. Such a layer aggregates the features of the last convolutional layer through spatial pooling, where the size of the pooling regions is dependent on the size of the input. Inspired by this, we propose to also consider adapting the network models of section 3.3.1 according to the above SPP-architecture. This can be easily achieved for all the considered models (e.g., see Fig. 3.4a for an example with a siamese model).

3.4 Learning

Optimization. We train all models in strongly supervised manner. We use a hinge-based loss term and squared l_2-norm regularization that leads to the following learning objective function

$$\min_w \frac{\lambda}{2}\|w\|_2 + \sum_{i=1}^{N} \max(0, 1 - y_i o_i^{net}) \ , \tag{3.7}$$

where w are the weights of the neural network, o_i^{net} is the network output for the i-th training sample, and $y_i \in \{-1, 1\}$ the corresponding label (with -1 and 1 denoting a non-matching and a matching pair, respectively).

ASGD with constant learning rate 1.0, momentum 0.9 and weight decay $\lambda = 0.0005$ is used to train the models. We find that ASGD is not essential and standard SDG can be used as well, however ASGD achieves slightly better results. Training is done in mini-batches of size 128. Weights are initialized randomly and all models are trained from scratch. We also find that batch normalization Ioffe and

Szegedy (2015) can be used to speed up training of siamese networks. In this case batch statistics should be computed across examples of both siamese branches, otherwise they produce very different results and training diverges.

Data Augmentation and preprocessing. To combat overfitting we augment training data by flipping both patches in pairs horizontally and vertically and rotating to 90, 180, 270 degrees. As we don't notice overfitting while training in such manner we train models for a certain number of iterations, usually for 2 days, and then test performance on test set.

Training dataset size allows us to store all the images directly in GPU memory and very efficiently retrieve patch pairs during training. Images are augmented "on-the fly". We use Titan GPU in Torch Collobert *et al.* (2011) and convolution routines are taken from cudnn library. Our NCC layer implementation is slightly slower than GEMM version of convolution on GPU, and significantly slower than cudnn direct convolutions. NCC could be significantly faster if implemented directly similar to cudnn or using FFT similar to Vasilache *et al.* (2014).

Our siamese descriptors on GPU are just 2 times slower than computing SIFT descriptors on CPU and 2 times faster than Imagenet descriptors on GPU according to Fischer *et al.* (2014).

3.5 Experimental results

We applied our models to a variety of problems and datasets. In the following we report extensive results, and also provide comparisons with the state-of-the-art.

3.5.1 Local image patches benchmark

For the first evaluation of our models, we used the standard benchmark dataset from Brown *et al.* (2011) that consists of three subsets, Yosemite, Notre Dame, and Liberty, each of which contains more than 450,000 image patches (64 x 64 pixels) sampled around Difference of Gaussians feature points. The patches are scale and orientation normalized. Each of the subsets was generated using actual 3D correspondences obtained via multi-view stereo depth maps. These maps were used to produce 500,000 ground-truth feature pairs for each dataset, with equal number of positive (correct) and negative (incorrect) matches.

train	test	2ch-ncc-avg	2ch-2stream	2ch-avg	2ch-deep	2ch	Simonyan *et al.* (2014)
#parameters		0.97M	2.35M	0.97M	1.08M	0.91M	-
Yos	ND	-	2.11	1.83	2.52	3.05	6.82
Yos	Lib	-	7.20	6.55	7.40	8.59	14.58
ND	Yos	**4.05**	4.09	5.13	4.38	6.04	10.08
ND	Lib	**3.63**	4.85	5.03	4.55	6.05	12.42
Lib	Yos	**4.24**	5.00	5.00	4.75	7.00	11.18
Lib	ND	**1.04**	1.90	1.70	2.01	3.03	7.22
mean		**3.24**	4.19	4.21	4.27	5.63	10.38
mean(1,4)		**3.84**	4.56	4.64	4.71	5.93	10.98

Table 3.1: FPR95 of 2-channel models on the "local image patches" benchmark. The models architecture is as follows: (i) `2ch-2stream` consists of two branches $\mathrm{C}(96,5,1)$-$\mathrm{P}(2,2)$-$\mathrm{C}(96,3,1)$-$\mathrm{P}(2,2)$-$\mathrm{C}(192,3,1)$-$\mathrm{C}(192,3,1)$, one for central and one for surround parts, followed by $\mathrm{F}(768)$-ReLU-$\mathrm{F}(1)$ (ii) `2ch-ncc-avg`: $\mathrm{NCC}(96,7,3)$-$\mathrm{P}(2,2)$-$\mathrm{NCC}(192,5,1)$-$\mathrm{NCC}(256,3,1)$-$\mathrm{NCC}(256,1,1)$-$\mathrm{A}(2,2)$-$\mathrm{F}(1)$ (iii) `2ch-deep`: $\mathrm{C}(96,4,3)$-$\mathrm{Stack}(96)$-$\mathrm{P}(2,2)$-$\mathrm{Stack}(192)$-$\mathrm{F}(1)$, where $\mathrm{Stack}(n)$=$\mathrm{C}(n,3,1)$-$\mathrm{C}(n,3,1)$-$\mathrm{C}(n,3,1)$. (iv) `2ch-avg`: $\mathrm{C}(96,7,3)$-$\mathrm{P}(2,2)$-$\mathrm{C}(192,5,1)$-$\mathrm{C}(256,3,1)$-$\mathrm{C}(256,1,1)$-$\mathrm{A}(2,2)$-$\mathrm{F}(1)$ (v) `2ch`: $\mathrm{C}(96,7,3)$-$\mathrm{P}(2,2)$-$\mathrm{C}(192,5,1)$-$\mathrm{P}(2,2)$-$\mathrm{C}(256,3,1)$-$\mathrm{C}(256,1,1)$-$\mathrm{F}(1)$ The shorthand notation used was the following: $\mathrm{C}(n,k,s)$ is a convolutional layer with n filters of spatial size $k \times k$ applied with stride s followed by ReLU; $\mathrm{NCC}(n,k,s)$ is NCC-layer with the same definition as $\mathrm{C}(n,k,s)$; $\mathrm{P}(k,s)$ is a max-pooling layer of size $k \times k$ applied with stride s; $\mathrm{A}(k,s)$ is avg-pooling with the same definition as max-pooling; $\mathrm{F}(n)$ denotes a fully connected linear layer with n output units.

For evaluating our models, we use the evaluation protocol of Brown *et al.* (2011) and generate ROC curves by thresholding the distance between feature pairs in the descriptor space. We report the false positive rate at 95% recall (FPR95) on each of the six combinations of training and test sets, as well as the mean across all combinations. We also report the mean, denoted as $\mathrm{mean}(1,4)$, for only those 4 combinations that were used in Boix *et al.* (2013), Brown *et al.* (2011) (in which case training takes place on Yosemite or Notre Dame, but not Liberty).

Table 3.1 and table 3.2 reports the performance of several models, and also details their architecture (we have also experimented with smaller kernels, less max-pooling layers, as well as adding normalizations, without noticing any significant improvement in performance). We briefly summarize some of the conclusions that can be drawn from these tables.

2-channel networks. A first important conclusion is that 2-channel-based architectures (e.g., `2ch`, `2ch-deep`, `2ch-2stream`) exhibit clearly the best performance among all models. This is something that indicates that it is important to *jointly* use information from both patches right from the *first* layer of the network. `2ch-2stream` network was the top-performing among convolutional networks on this dataset, with `2ch-deep` following closely (this verifies the importance of multi-resolution information during matching and that also increasing the network depth helps). In fact, `2ch-2stream` managed

train	test	siam	siam l_2(256d)	pseudo-siam	pseudo-siam l_2(256d)	siam-2stream	siam-2stream l_2(512d)	Simonyan *et al.* (2014)
#parameters		1.17M	0.9M	2.08M	0.9M	5.85M	2.4M	-
Yos	ND	5.75	8.38	5.44	8.95	5.29	5.58	6.82
Yos	Lib	13.48	17.25	12.64	18.37	11.51	12.84	14.58
ND	Yos	13.23	15.89	10.35	15.62	10.44	13.02	10.08
ND	Lib	8.77	13.24	12.87	16.58	6.45	8.79	12.42
Lib	Yos	14.89	19.91	12.50	17.83	9.02	13.24	11.18
Lib	ND	4.33	6.01	3.93	6.58	3.05	4.54	7.22
mean		10.07	13.45	9.62	13.99	7.63	9.67	10.38
mean(1,4)		10.31	13.69	10.33	14.88	8.42	10.06	10.98

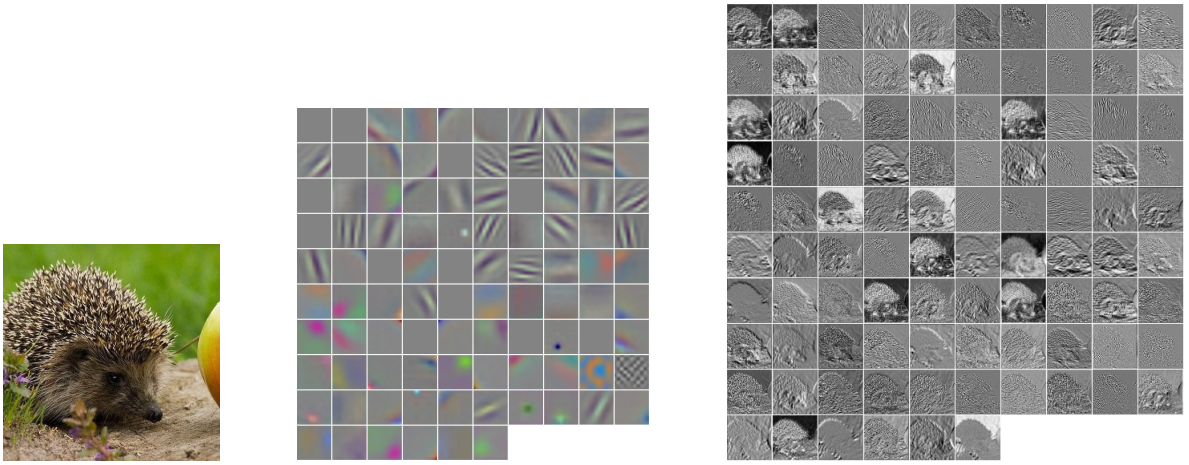
Table 3.2: Performance of siamese models on the "local image patches" benchmark. The models architecture is as follows: (i) `siam` has two branches $\mathrm{C}(96,7,3)$-$\mathrm{P}(2,2)$-$\mathrm{C}(192,5,1)$-$\mathrm{P}(2,2)$-$\mathrm{C}(256,3,1)$ and decision layer $\mathrm{F}(512)$-ReLU-$\mathrm{F}(1)$ (ii) `siam`-l_2 reduces to a single branch of `siam` (iii) `pseudo-siam` is uncoupled version of `siam` (iv) `pseudo-siam`-l_2 reduces to a single branch of `pseudo-siam` (v) `siam-2stream` has 4 branches $\mathrm{C}(96,4,2)$-$\mathrm{P}(2,2)$-$\mathrm{C}(192,3,1)$-$\mathrm{C}(256,3,1)$-$\mathrm{C}(256,3,1)$ (coupled in pairs for central and surround streams), and decision layer $\mathrm{F}(512)$-ReLU-$\mathrm{F}(1)$ (vi) `siam-2stream`-l_2 consists of one central and one surround branch of `siam-2stream`. The shorthand notation used was the following: $\mathrm{C}(n,k,s)$ is a convolutional layer with n filters of spatial size $k \times k$ applied with stride s followed by ReLU, $\mathrm{P}(k,s)$ is a max-pooling layer of size $k \times k$ applied with stride s, and $\mathrm{F}(n)$ denotes a fully connected linear layer with n output units.

to outperform the previous state-of-the-art by a large margin, achieving 2.45 times better score than Simonyan *et al.* (2014)! The difference with SIFT was even larger, with our model giving 6.65 times better score in this case (SIFT score on `mean(1,4)` was 31.2 according to Brown *et al.* (2011)).

To construct `2ch-avg` from `2ch` we remove the second max-pooling and insert average-pooling with the same kernel size and stride after the last convolutional layer. As expected, `2ch-avg` shows better results than `2ch` with less parameters. It is only slightly worse than `2ch-2stream` without utilizing 2-stream architecture, which should further improve performance.

NCC-networks. We used `2ch-avg` architecture to construct NCC-network `2ch-ncc-avg`, which outperformed all other networks by a significant margin. As NCC-networks improve significantly the results with our best architecture, we believe that's enough to show their supremacy in patch comparing tasks. For the goal of achieving better convolutional descriptors for l_2 matching we expect NCC-networks to achieve superior performance when trained using recently proposed triplet training Balntas *et al.* (2016) or global loss functions G *et al.* (2016). On fig. 3.8c we show first and second channel filters of `2ch-ncc-avg` network.

To show that the proposed NCC-networks are general and can be applied in other tasks, we trained AlexNet with NCC layers instead of convolutional on ImageNet-2012 image classification dataset.



(a) Input image (b) Filters of the first NCC-layer (c) Outputs of the first NCC-layer

Figure 3.5: Visualization of the filters and outputs of the first NCC-layer of NCC-AlexNet trained on ImageNet.

NCC-AlexNet gives 1% higher both top1 and top5 single crop accuracy than convolutional AlexNet (both networks trained with batch normalization), improving top1 from 56.2% to 57.2%. For reference we provide learnt filters of the first layer visualization with the corresponding outputs of the first layer in figure 3.5. The filters are very similar to those learnt by baseline convolutional AlexNet. Both networks were modified to use batch normalization and dropout was removed. Using batch normalization was essential to obtaining improvement over baseline, but not required for making network train with proper initialization. We speculate that in this case batch normalization acts as a per-layer learning rate to normalized network that doesn't have a way to scale itself otherwise, thus improving training process. With batch normalization NCC-layers are initialized with normal distribution. We plan to investigate these intriguing findings in future work.

Overall, the architectures of 2-channel networks are the following:

- `2ch-2stream` consists of two branches $\mathrm{C}(96,5,1)$-$\mathrm{P}(2,2)$-$\mathrm{C}(96,3,1)$-$\mathrm{P}(2,2)$-$\mathrm{C}(192,3,1)$-$\mathrm{C}(192,3,1)$, one for central and one for surround parts, followed by $\mathrm{F}(768)$-ReLU-$\mathrm{F}(1)$
- `2ch-deep`: $\mathrm{C}(96,4,3)$-$\mathrm{Stack}(96)$-$\mathrm{P}(2,2)$-$\mathrm{Stack}(192)$-$\mathrm{F}(1)$, where $\mathrm{Stack}(n)$=$\mathrm{C}(n,3,1)$-$\mathrm{C}(n,3,1)$-$\mathrm{C}(n,3,1)$
- `2ch`: $\mathrm{C}(96,7,3)$-$\mathrm{P}(2,2)$-$\mathrm{C}(192,5,1)$-$\mathrm{P}(2,2)$-$\mathrm{C}(256,3,1)$-$\mathrm{C}(256,1,1)$-$\mathrm{F}(1)$
- `2ch-avg`: $\mathrm{C}(96,7,3)$-$\mathrm{P}(2,2)$-$\mathrm{C}(192,5,1)$-$\mathrm{C}(256,3,1)$-$\mathrm{C}(256,1,1)$-$\mathrm{A}(2,2)$-$\mathrm{F}(1)$
- `2ch-ncc-avg`: $\mathrm{NCC}(96,7,3)$-$\mathrm{P}(2,2)$-$\mathrm{NCC}(192,5,1)$-$\mathrm{NCC}(256,3,1)$-$\mathrm{NCC}(256,1,1)$-$\mathrm{A}(2,2)$-$\mathrm{F}(1)$

The shorthand notation used was the following: $\mathrm{C}(n, k, s)$ is a convolutional layer with n filters of spatial size $k \times k$ applied with stride s followed by ReLU, $\mathrm{P}(k, s)$ is a max-pooling layer of size $k \times k$ applied with stride s, and $\mathrm{F}(n)$ denotes a fully connected linear layer with n output units.

Siamese/pseudo-siamese networks. Regarding siamese-based architectures, these too manage to achieve better performance than existing state-of-the-art systems. This is quite interesting because, e.g., none of these siamese networks tries to learn the shape, size or placement of the pooling regions (like, e.g., Simonyan *et al.* (2014); Brown *et al.* (2011) do), but instead utilizes just standard max-pooling layers. Among the siamese models, the two-stream network (`siam-2stream`) had the best performance, verifying once more the importance of multi-resolution information when it comes to comparing image patches. Furthermore, the pseudo-siamese network (`pseudo-siam`) was better than the corresponding siamese one (`siam`).

We also conducted additional experiments, in which we tested the performance of siamese models when their top decision layer is replaced with the l_2 Euclidean distance of the two convolutional descriptors produced by the two branches of the network (denoted with the suffix l_2 in the name). In this case, prior to applying the Euclidean distance, the descriptors are l_2-normalized (we also tested l_1 normalization). For pseudo-siamese only one branch was used to extract descriptors. As expected, in this case the two-stream network (`siam-2stream-`l_2) computes better distances than the siamese network (`siam-`l_2), which, in turn, computes better distances than the pseudo-siamese model (`pseudo-siam-`l_2). In fact, the `siam-2stream-`l_2 network manages to outperform even the previous state-of-the-art descriptor Simonyan *et al.* (2014), which is quite surprising given that these siamese models have never been trained using l_2 distances.

The pseudo-siam network has two uncoupled branches which make it asymmetric. It is possible to make its decision symmetric by taking the sum of decisions from both possible combinations of patches in pair. Let P_1 and P_2 be the patches in pair and $o(P_1, P_2)$ - network's decision on these patches. Then the symmetric decision is defined as:

$$o_s(P_1, P_2) = o(P_1, P_2) + o(P_2, P_1) \tag{3.8}$$

In table 3.3 we show the results of evaluation of the above decision function. It's mean FPR95 over all dataset combinations is 9.11, which is by 0.63 better than a single asymmetric decision result and by 0.96 better than a result of siam network.

Overall, siamese models have the following architectures:

		$o(P_1, P_2)$	$o(P_1, P_2) + o(P_2, P_1)$
Yos	ND	5.44	4.82
Yos	Lib	12.64	11.79
ND	Yos	13.61	13.25
ND	Lib	10.35	9.99
Lib	Yos	12.50	11.44
Lib	ND	3.93	3.37
mean		9.74	9.11
mean(1,4)		10.51	9.96

Table 3.3: Results of pseudo-siam network with symmetric decision function evaluation

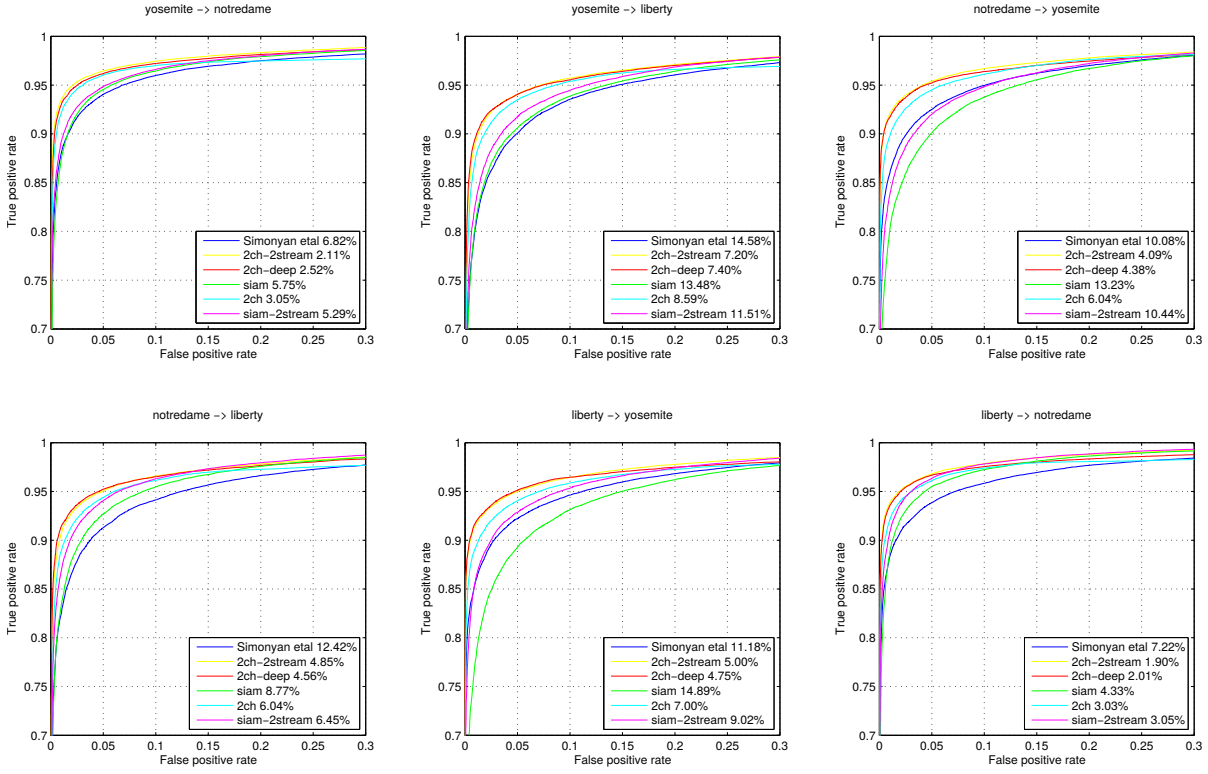


Figure 3.6: ROC curves for various models (including the state-of-the-art descriptor Simonyan *et al.* (2014)) on the local image patches benchmark. Numbers in the legends are corresponding FPR95 values

- `siam` has two branches $\mathrm{C}(96, 7, 3)$-$\mathrm{P}(2, 2)$-$\mathrm{C}(192, 5, 1)$-$\mathrm{P}(2, 2)$-$\mathrm{C}(256, 3, 1)$ and decision layer $\mathrm{F}(512)$-ReLU-$\mathrm{F}(1)$
- `siam`-l_2 reduces to a single branch of `siam`
- `pseudo-siam` is uncoupled version of `siam`
- `pseudo-siam`-l_2 reduces to a single branch of `pseudo-siam`
- `siam-2stream` has 4 branches $\mathrm{C}(96, 4, 2)$-$\mathrm{P}(2, 2)$-$\mathrm{C}(192, 3, 1)$-$\mathrm{C}(256, 3, 1)$-$\mathrm{C}(256, 3, 1)$ (coupled in pairs for central and surround streams), and decision layer $\mathrm{F}(512)$-ReLU-$\mathrm{F}(1)$

	$\text{conv3}_{(3456)}$	$\text{conv4}_{(3456)}$	$\text{conv5}_{(2304)}$
Notredame	12.22	9.64	19.384
Liberty	16.25	14.26	21.592
Yosemite	33.25	30.22	43.262
mean	20.57	17.98	28.08

Table 3.4: FPR95 of ImageNet model features (dimensionality of each feature appears as subscript). We feed AlexNet with resized grayscale patches, and extract features from different convolutional layers.

- `siam-2stream-`l_2 consists of one central and one surround branch of `siam-2stream`

We recall again that the following shorthand notation was used: $\mathrm{C}(n, k, s)$ is a convolutional layer with n filters of spatial size $k \times k$ applied with stride s followed by ReLU, $\mathrm{P}(k, s)$ is a max-pooling layer of size $k \times k$ applied with stride s, and $\mathrm{F}(n)$ denotes a fully connected linear layer with n output units.

For a more detailed comparison of the various models, we provide the corresponding ROC curves in Fig. 3.6. Furthermore, we show in Table 3.4 the performance of imagenet-trained CNN features (these were l_2-normalized to improve results). Among these, `conv4` gives the best FPR95 score, which is equal to 17.98. This makes it better than SIFT but still much worse than our models. We provide here a more detailed quantitative comparison of l_2-decision networks (i.e., where we use l_2 distance to compare descriptors at test time). To that end, we show the corresponding ROC curves in figure 3.7, comparing also with the state-of-the-art method Simonyan *et al.* (2014). As can be observed, the siam-2stream-l_2 model exhibits the best performance on all datasets combinations except when being tested on Yosemite.

Fig. 3.8a displays the filters of the first convolutional layer learnt by the siamese network. Furthermore, Fig. 3.8b shows the subset of first and second channel first layer filters of the 2-channel network `2ch`. It is worth mentioning that corresponding first and second channel parts look like being negative to each other, which basically means that the network has learned to compute differences of features between the two patches (note, though, that not all first layer filters of `2ch` look like this). Last, we show in Fig. 3.9 some top ranking false and correct matches as computed by the `2ch-deep` network. We observe that false matches could be easily mistaken even by a human (notice, for instance, how similar the two patches in false positive examples look like).

For the rest of the experiments, we note that we use models trained on the Liberty dataset.

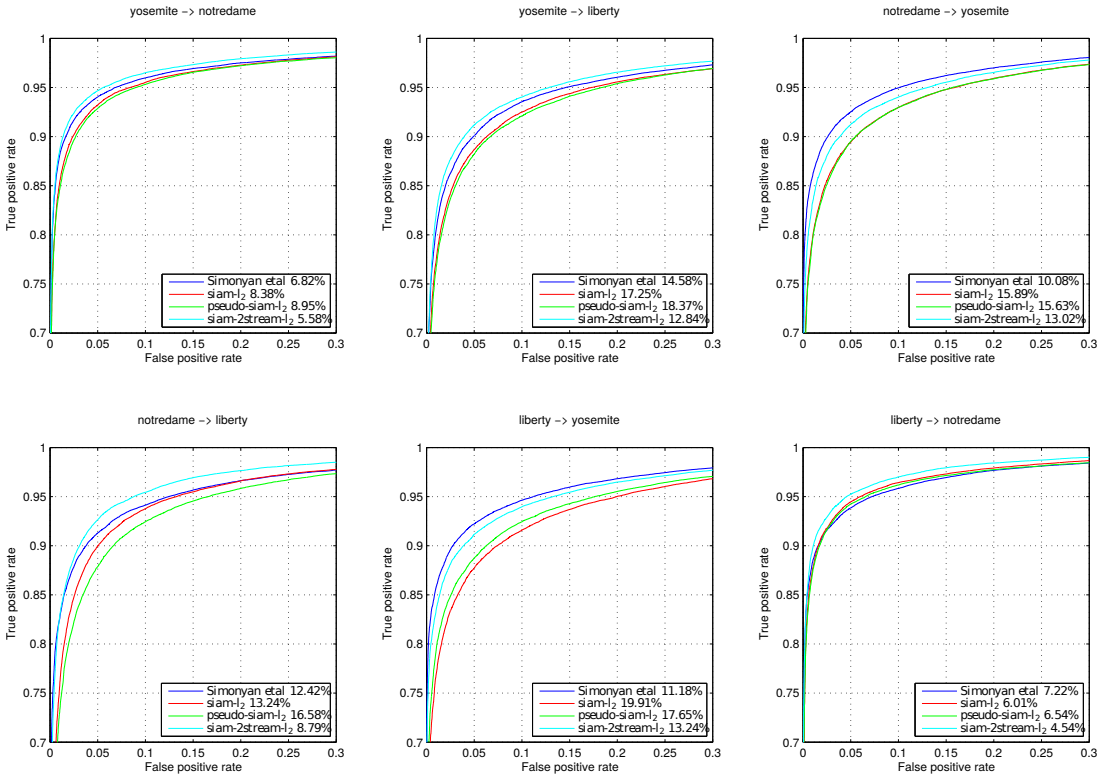


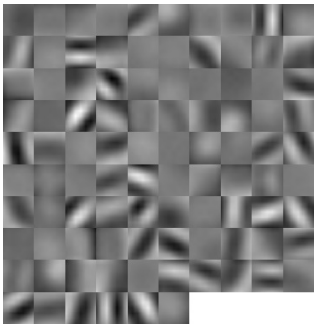
Figure 3.7: ROC curves of l_2 networks. siam-2stream-l_2 shows the best performance on 4 out of 6 combinations of sequences

3.5.2 Wide baseline stereo evaluation

For this evaluation we chose the dataset by Strecha *et al.* Strecha *et al.* (2008), which contains several image sequences with ground truth homographies and laser-scanned depthmaps. We used "fountain" (fig. 3.10) and "herzjesu" (fig. 3.11) sequences to produce 6 and 5 rectified stereo pairs respectively. Baselines in both sequences we chose are increasing with each image making matching more difficult. Our goal was to show that a photometric cost computed with neural network competes favorably against costs produced by a state-of-the-art hand-crafted feature descriptor, so we chose to compare with DAISY Tola *et al.* (2008).

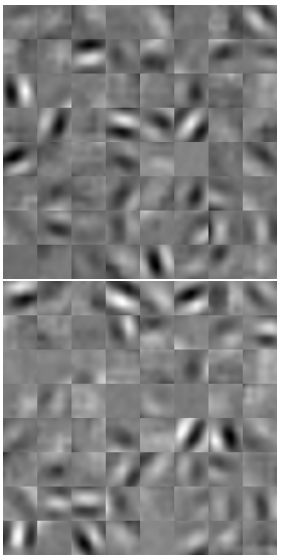
Since our focus was not on efficiency, we used an unoptimized pipeline for computing the photometric costs. More specifically, for 2-channel networks we used a brute-force approach, where we extract patches on corresponding epipolar lines with subpixel estimation, construct batches (containing a patch from the left image I_1 and all patches on the corresponding epipolar line from the right image I_2) and compute network outputs, resulting in the cost:

$$C(\mathbf{p}, d) = -o^{net}(I_1(\mathbf{p}), I_2(\mathbf{p} + d)) \tag{3.9}$$

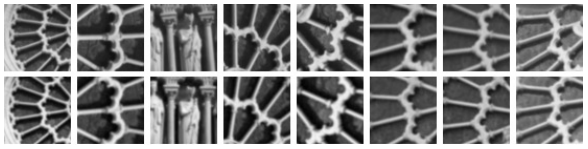


(a) Filters of the first convolutional layer of `siam` network.

(b) Rows correspond to first layer filters from `2ch` network (only a subset shown), depicting first and second channel filters.



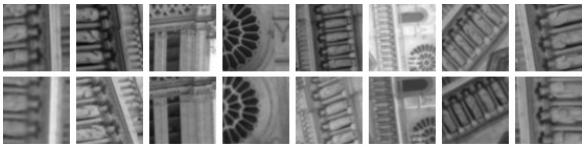
(c) First and second channel filters of the first NCC-layer of `2ch-ncc-avg` network



(a) true positives

(b) false negatives

(c) true negatives



(d) false positives

Figure 3.9: Top-ranking false and true matches by `2ch-deep`.

Here, $I(\mathbf{p})$ denotes a neighbourhood intensity matrix around a pixel $\mathbf{p}$, $o^{net}(P_1, P_2)$ is the output of the neural network given a pair of patches P_1 and P_2, and d is the distance between points on epipolar line.

For siamese-type networks, we compute descriptors for each pixel in both images once and then match them with decision top layer or l_2 distance. In the first case the formula for photometric cost is the following:

$$C(\mathbf{p}, d) = -o^{top}(D_1(I_1(\mathbf{p})), D_2(I_2(\mathbf{p} + d))) \tag{3.10}$$

where o^{top} is output of the top decision layer, and D_1, D_2 are outputs of branches of the siamese or pseudo-siamese network, i.e. descriptors (in case of siamese network $D_1 = D_2$). For l_2 matching, it holds:

$$C(\mathbf{p}, d) = \|D_1(I_1(\mathbf{p})) - D_2(I_2(\mathbf{p} + d))\|_2 \tag{3.11}$$


(a) Image 0002


(b) Image 0003


(c) Image 0004


(d) Image 0005


(e) Image 0006


(f) Image 0007


(g) Image 0008

Figure 3.10: Images from "fountain" dataset. We use images 0002-0008 to generate 6 rectified stereo pairs against image 0003


(a) Image 0000


(b) Image 0001


(c) Image 0002


(d) Image 0003


(e) Image 0004


(f) Image 0005

Figure 3.11: Images from "herzjesu" dataset. We use images 0000-0005 to generate 5 stereo pairs against image 0005.

It is worth noting that all costs above can be computed a lot more efficiently using speed optimizations similar with Zbontar and LeCun (2015). This essentially means treating all fully connected layers as 1×1 convolutions, computing branches of siamese network only once, and furthermore computing the outputs of these branches as well as the final outputs of the network at all locations using a number of forward passes on full images. For a 2-channel architecture such an approach of computing the photometric costs would only require feeding the network with $s^2 \cdot d_{\max}$ full 2-channel images of size equal to the input image pair, where s is the stride at the first layer of the network and $d_{\max}$ is the maximum disparity. This scenario might be interesting for real-time stereo applications, where the pairs could be packed in one batch of images and processed very efficiently in fully feed-forward manner without involving any descriptor matching. Modern GPUs are exceptionally efficient in this setting.

Once computed, the photometric costs are subsequently used as unary terms in the following pairwise MRF energy

$$E(\{d_p\}) = \sum_p C(\mathbf{p}, d_p) + \sum_{(p,q)\in\mathcal{E}} (\lambda_1 + \lambda_2 e^{-\frac{\|\nabla I_1(\mathbf{p})\|_2}{\sigma^2}}) \cdot |d_p - d_q|\,,$$

minimized using algorithm Conejo *et al.* (2014) based on FastPD Komodakis *et al.* (2007) (we set $\lambda_1 = 0.01$, $\lambda_2 = 0.2$, $\sigma = 7$ and $\mathcal{E}$ is a 4-connected grid).

We show in fig. 3.14 and fig. 3.15 (also close-up views in fig. 3.17 and fig. 3.19) some qualitative results in terms of computed depth maps (with and without global optimization) for the "fountain" image set. Global MRF optimization results visually verify that photometric cost computed with neural network is much more robust than with hand-crafted features, as well as the high quality of the depth maps produced by 2-channel architectures. Results without global optimization also show that the estimated depth maps contain much more fine details than DAISY. They may exhibit a very sparse set of errors for the case of siamese-based networks, but these errors can be very easily eliminated during global optimization. Close-up view reveals the ability of our networks to capture small details, unavailable to DAISY. We especially note very good `siam-2stream-`l_2's performance, confirming quantitative results.

Fig. 3.12 and fig. 3.13 also shows a quantitative comparison, focusing in this case on siamese-based models as they are more efficient. The first plot of that figure shows (for a single stereo pair) the distribution of deviations from the ground truth across all range of error thresholds (expressed here as a fraction of the scene's depth range). Furthermore, the other plots of the same figure summarize the corresponding distributions of errors for the six stereo pairs of increasing baseline (in this case we also show separately the error distributions when only unoccluded pixels are taken into account). The error thresholds were set to 3 and 5 pixels in these plots (note that the maximum disparity is around 500 pixels in the largest baseline). As can be seen, all siamese models perform much better than DAISY across all error thresholds and all baseline distances (e.g., notice the difference in the curves of the corresponding plots).

3.5.3 Local descriptors performance evaluation

We also test our networks on Mikolajczyk dataset for local descriptors evaluation Mikolajczyk and Schmid (2005). The dataset consists of 48 images in 8 sequences with camera viewpoint changes, blur, compression, lighting changes and zoom with gradually increasing amount of transformation. There are known ground truth homographies between the first and each other image in sequence.

Testing technique is the same as in Mikolajczyk and Schmid (2005). Briefly, to test a pair of images, detectors are applied to both images to extract keypoints. Following Fischer *et al.* (2014), we use MSER detector. The ellipses provided by detector are used to extract patches from input images. Ellipse size is magnified by a factor of 3 to include more context. Then, depending on the type of network, either

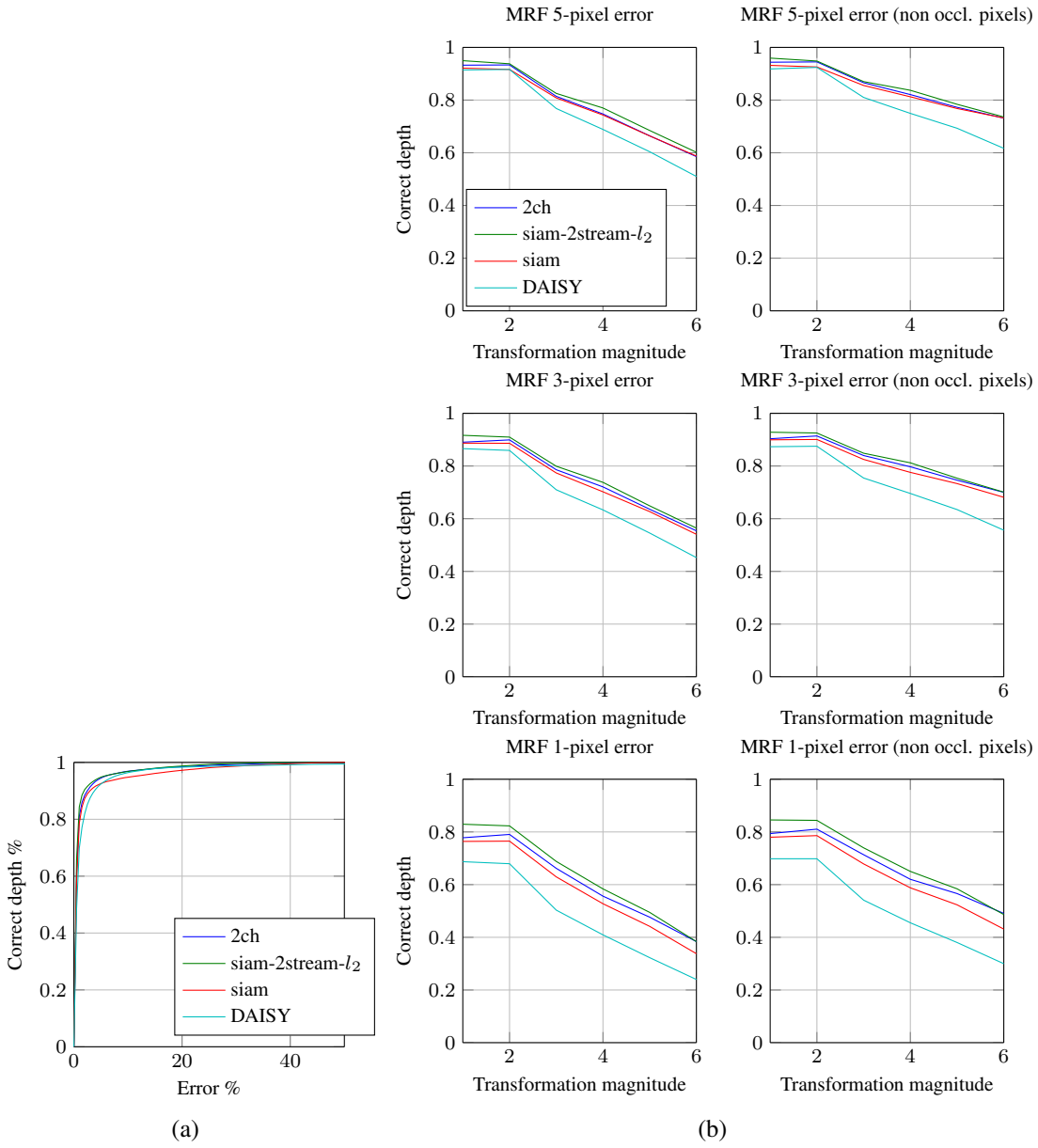


Figure 3.12: Quantitative comparison for wide baseline stereo evaluation on "fountain" dataset. (a) Distributions of deviations from the laser-scan data, expressed as a fraction of the scene's depth range of the second depth map in the sequence. (b) Distribution of errors for stereo pairs of increasing baseline (horizontal axis) both with and without taking into account occluded pixels (error thresholds were set equal to 5, 3 and 1 pixels in these plots - maximum disparity is around 500 pixels).

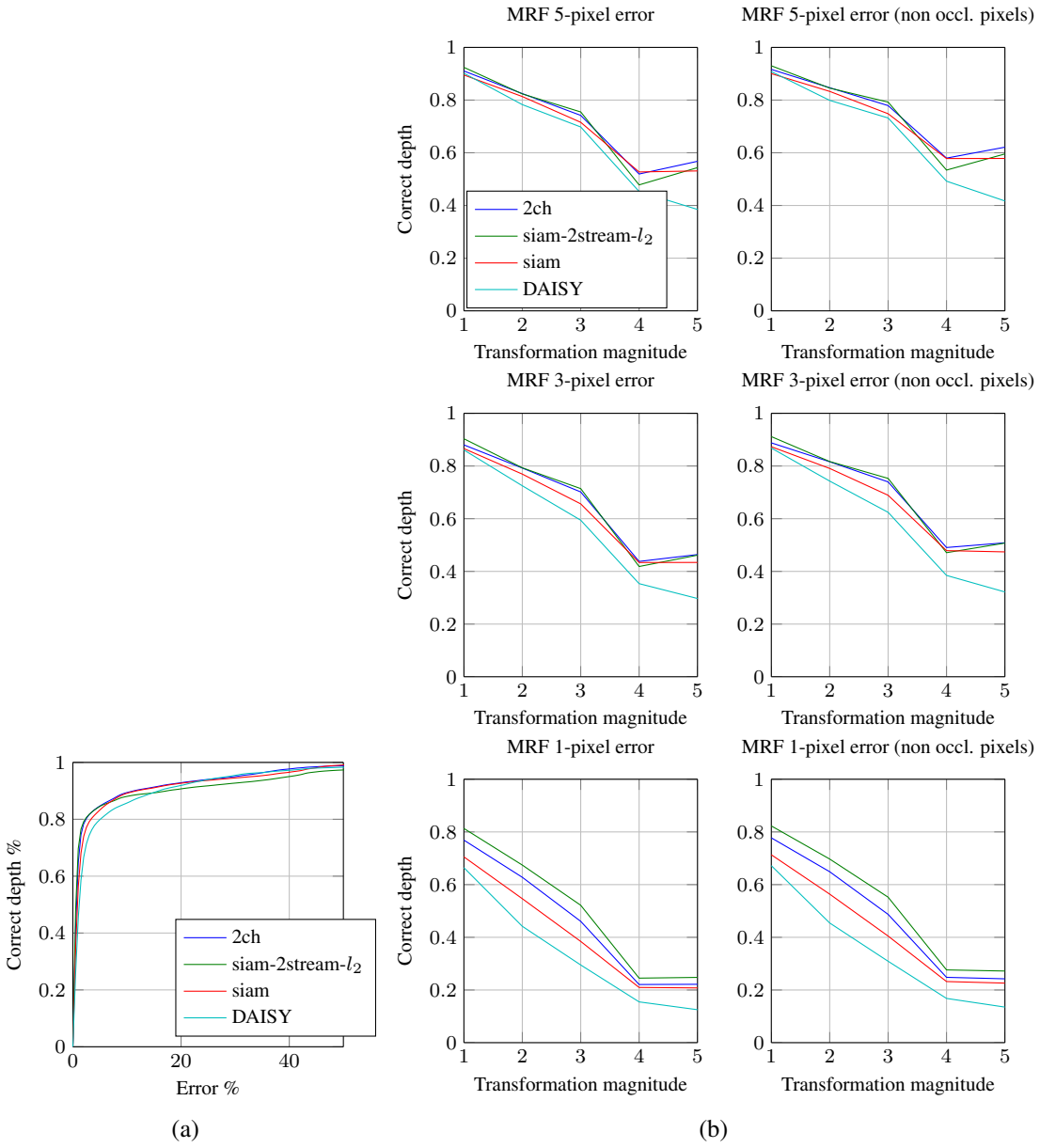


Figure 3.13: Quantitative comparison for wide baseline stereo on "herzjesu" dataset. (a) Distributions of deviations from the laser-scan data, expressed as a fraction of the scene's depth range of the second of the second depth map in the sequence. (b) Distribution of errors for stereo pairs of increasing baseline (horizontal axis) both with and without taking into account occluded pixels (error thresholds were set equal to 5, 3 and 1 pixels in these plots - maximum disparity is around 500 pixels).

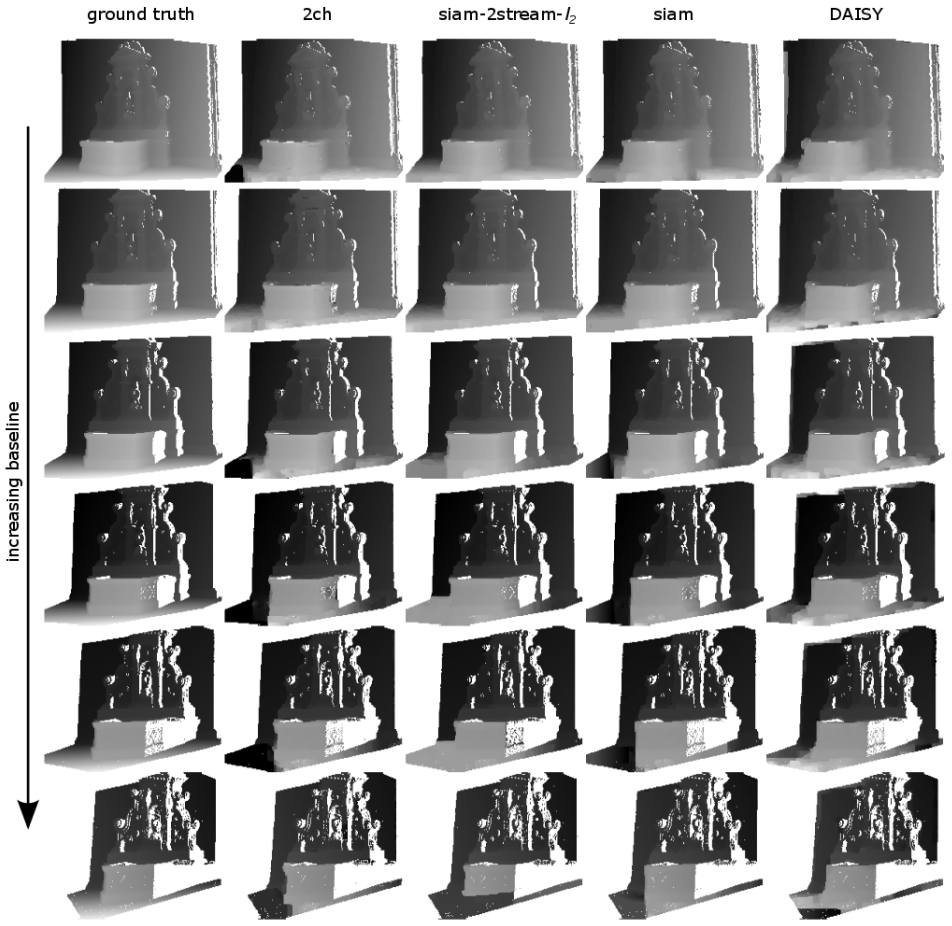


Figure 3.14: Qualitative comparison for wide baseline stereo evaluation on "fountain" dataset. From left to right column we show depth maps from ground truth, 2ch, siam-2stream-l_2, siam networks and DAISY. The baseline between stereo pairs increases from top to bottom. All depth maps were computed with MRF optimization, only non-occluded pixels are shown.

descriptors, meaning outputs of siamese or pseudo-siamese branches, are extracted, or all patch pairs are given to 2-channel network to assign a score.

A quantitative comparison on this dataset is shown for several models in Fig. 3.20, 3.21, 3.22, 3.23. Fig. 3.20 provides evaluation plots for all sequences from Mikolajczyk dataset Mikolajczyk and Schmid (2005). To compute the performance measure we extract elliptic regions of interest and corresponding image patches from both images using MSER detector. Minimal area size of detected ellipses set to 100. Next we compute the descriptors of all extracted patches and match all of them based on l_2 distance. A pair is a true positive if and only if the ellipse of the descriptor in the target image and the ground truth ellipse have an intersection over union that is greater than or equal to 0.6 (all other pairs are false positives). Based on this, a precision recall curve is computed and the area under this curve (average precision) is used as performance measure (mAP).

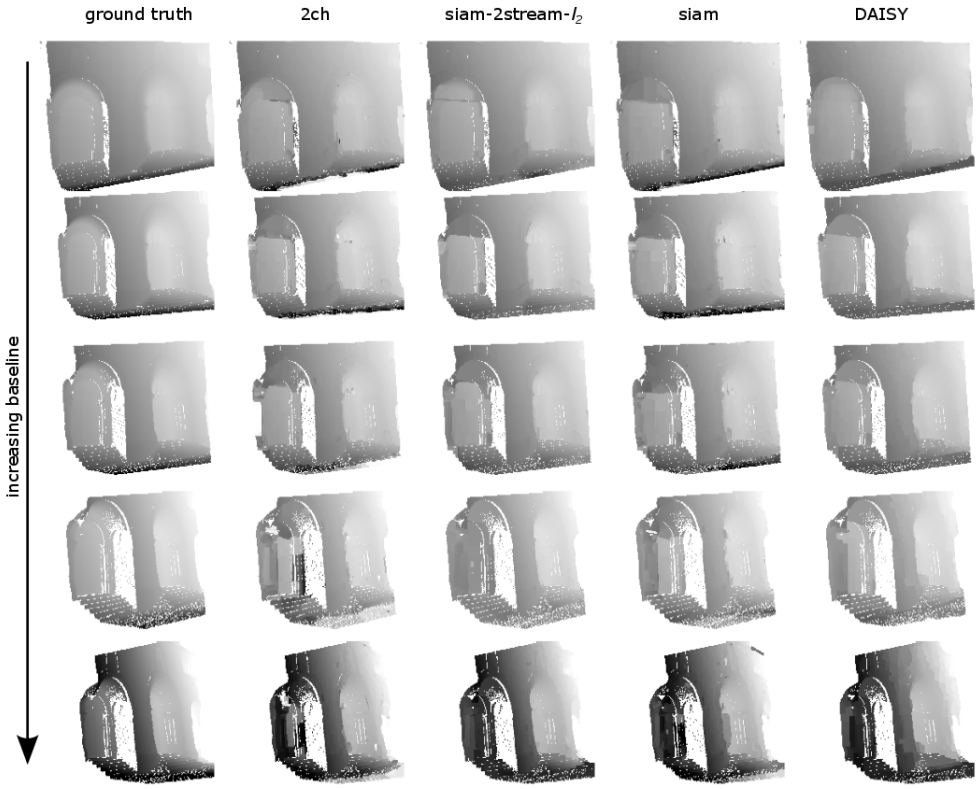
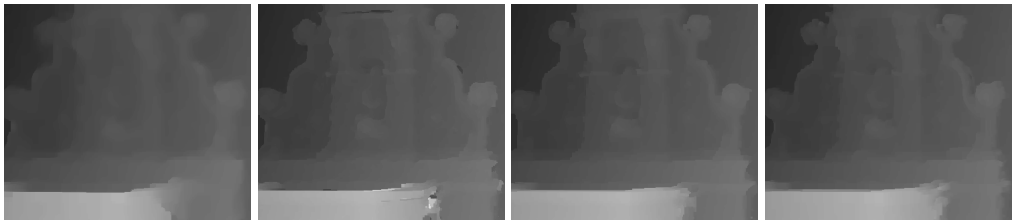


Figure 3.15: Qualitative comparison for wide baseline stereo evaluation on "herzjesu" dataset. From left to right column we show depth maps from ground truth, 2ch, siam-2stream-l_2, siam networks and DAISY. The baseline between stereo pairs increases from top to bottom. All depth maps were computed with MRF optimization, only non-occluded pixels are shown.

Here we also test the CNN network `siam-SPP-`l_2, which is an SPP-based siamese architecture (note that `siam-SPP` is same as `siam` but with the addition of two SPP layers - see also Fig. 3.4a). We used an inserted SPP layer that had a spatial dimension of 4×4. As can be seen, this provides a big boost in matching performance, suggesting the great utility of such an architecture when comparing image patches. Regarding the rest of the models, the observed results in Fig. 3.21 reconfirm the conclusions already drawn from previous experiments. Surprisingly, `2ch-ncc-avg` does worse than `2ch-2stream`. We also note again the very good performance of `siam-2stream-`l_2, which (although not trained with l_2 distances) is able to significantly outperform SIFT and to also match the performance of imagenet-trained features (using, though, a much lower dimensionality of 512).

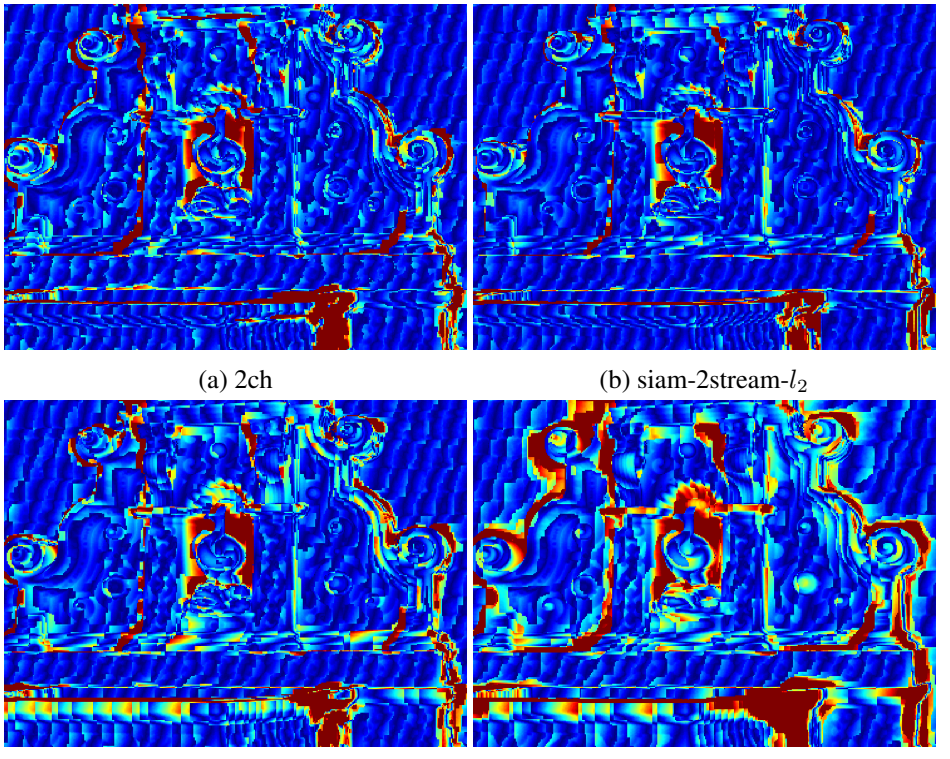
We also experimented with evaluating the performance of SPP-based networks when using SPP layers of different spatial sizes. Minimal area size of detected with MSER ellipses set to 100. The results



(a) DAISY (b) 2ch (c) siam-2stream-l_2 (d) siam

(e) Ground truth

Figure 3.16: Close-up views on wide-baseline stereo evaluation results on "fountain" dataset. All networks are better than DAISY, with `siam-2stream-`l_2 is better than `siam`, `2-ch` is comparable, but has some small artifacts.

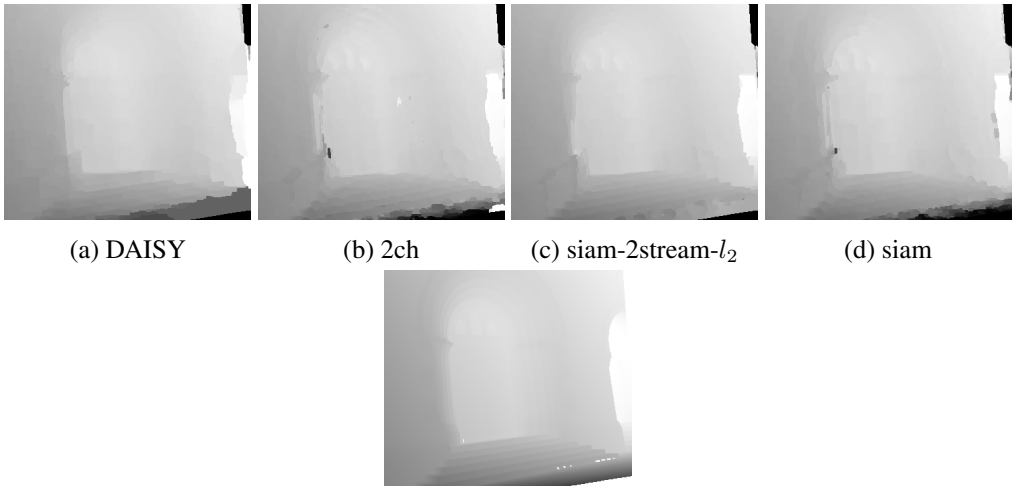


(a) 2ch (b) siam-2stream-l_2

(c) siam (d) DAISY

(e)

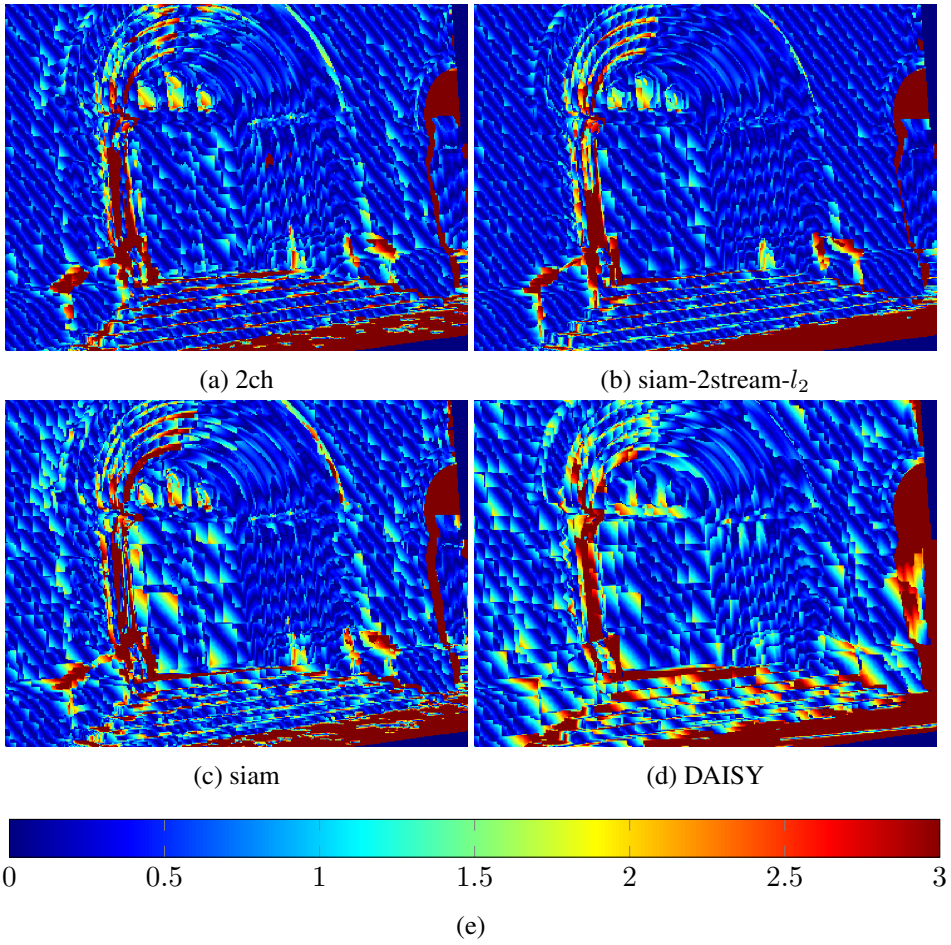
Figure 3.17: For the close-up views of fig. 3.16 we show thresholded absolute differences of ground truth depth map and estimated depth maps. Threshold is set to 3 pixels. All networks are better than DAISY.



(a) DAISY (b) 2ch (c) siam-2stream-l_2 (d) siam

(e) Ground truth

Figure 3.18: Close-up views on wide-baseline stereo evaluation results on "herzjesu" dataset. All networks are better than DAISY, with `siam-2stream-`l_2 is better than `siam`, `2-ch` is comparable, but has some small artifacts.



(a) 2ch (b) siam-2stream-l_2

(c) siam (d) DAISY

(e)

Figure 3.19: For the close-up views of fig. 3.18 we show thresholded absolute differences of ground truth depth map and estimated depth maps. Threshold is set to 3 pixels.

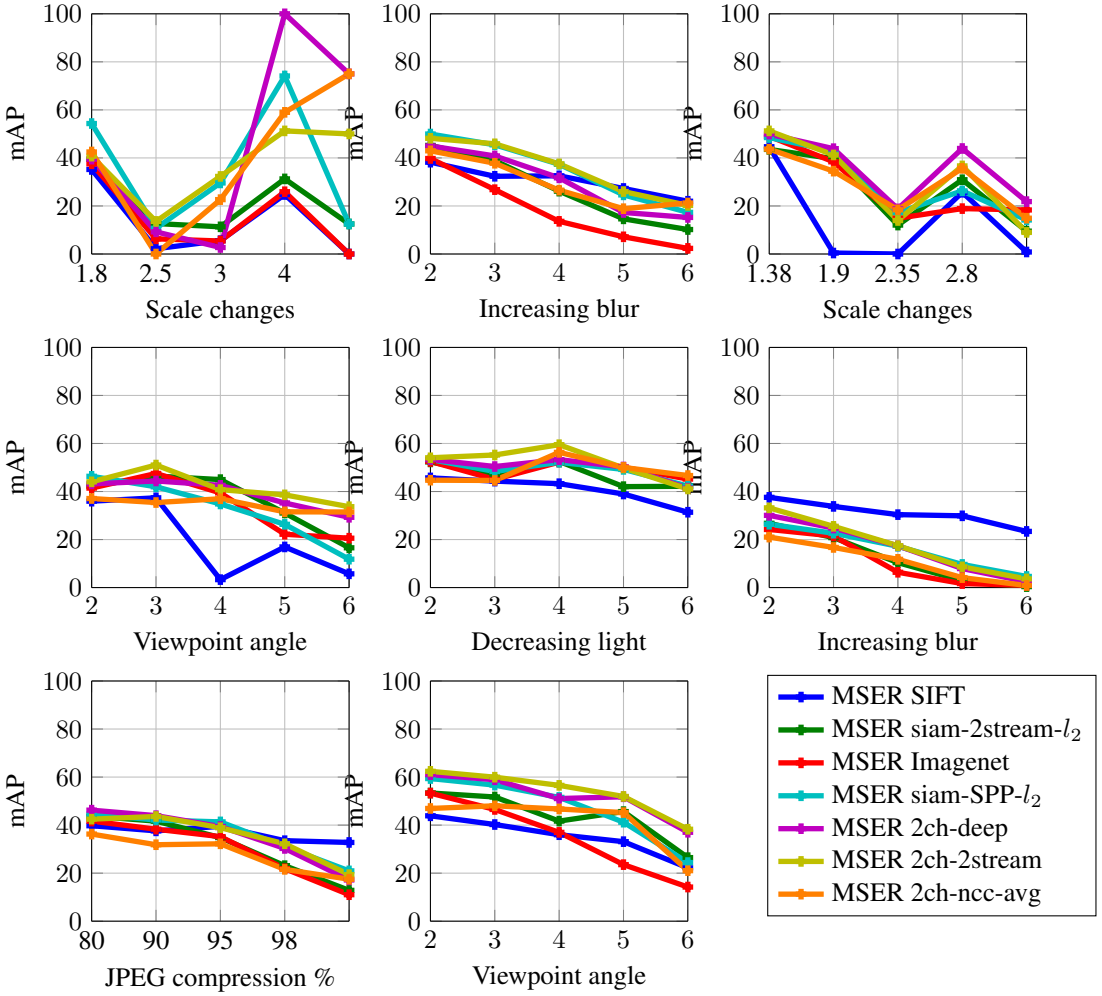


Figure 3.20: Evaluation plots of local descriptors on different datasets (i.e., with different transformations). Horizontal axis represents the transformation magnitude in each case.

in fig. 3.22 concern the model siam-SPP-l_2 (recall that siam-SPP is obtained using siam descriptors, with spatial max-pooling module inserted after the second convolutional layer). The input patches were rescaled such that $\min(\text{width}, \text{height}) > a$ where a is a minimal image size accepted by the network and were equal to 34, 40, 46 and 64 for 1×1, 2×2, 3×3 and 4×4 spatial pooling output sizes respectively. Fig. 3.23 shows average mAP of all datasets. The results show that increasing pooling output size consistently improves results. It has to be noted that increasing pooling output leads to increased dimensionality of the descriptor, for example, 4x4 output size produces $192 \times 4 \times 4 = 3072$ dimensional feature. SPP performance can improve even further, as no multiple aspect ratio patches were used during training (these appear only at test time).

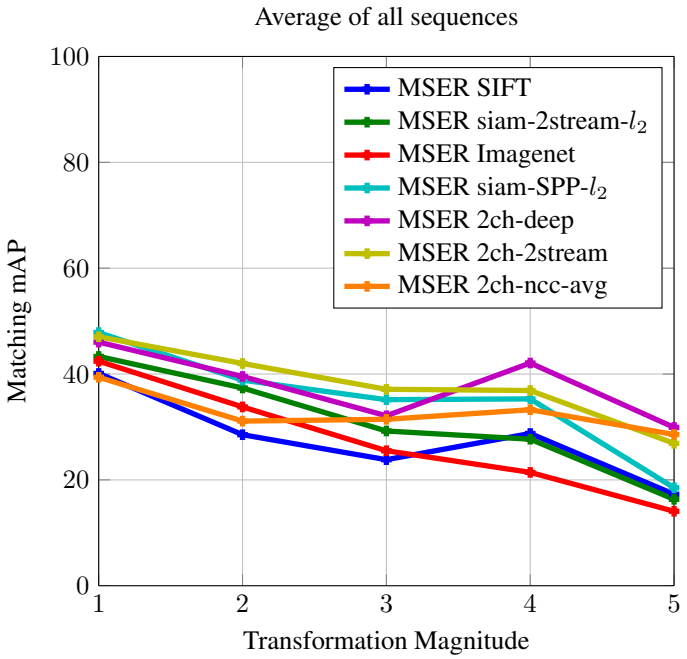


Figure 3.21: Overall evaluation of local descriptors showing the average performance over all datasets in Fig. 3.20.

3.6 Conclusions

In this chapter we showed how to learn directly from raw image pixels a general similarity function for patches, which is encoded in the form of a CNN model. To that end, we studied several neural network architectures that are specifically adapted to this task, and showed that they exhibit extremely good performance, significantly outperforming the state-of-the-art on several problems and benchmark datasets.

Among these architectures, we note that 2-channel-based ones were clearly the superior in terms of results. It is, therefore, worth investigating how to further accelerate the evaluation of these networks in the future. As for wide-baseline stereo, we proposed a fast promising approach to compute the stereo cost in batched fully feed-forward manner not involving descriptor matching.

Regarding siamese-based architectures, 2-stream multi-resolution models turned out to be extremely strong, providing always a significant boost in performance and verifying the importance of multi-resolution information when comparing patches. The same conclusion applies to SPP-based siamese networks, which also consistently improved the quality of results[1].

[1] In fact, SPP performance can improve even further, as no multiple aspect ratio patches were used during the training of SPP models (such patches appear only at test time).

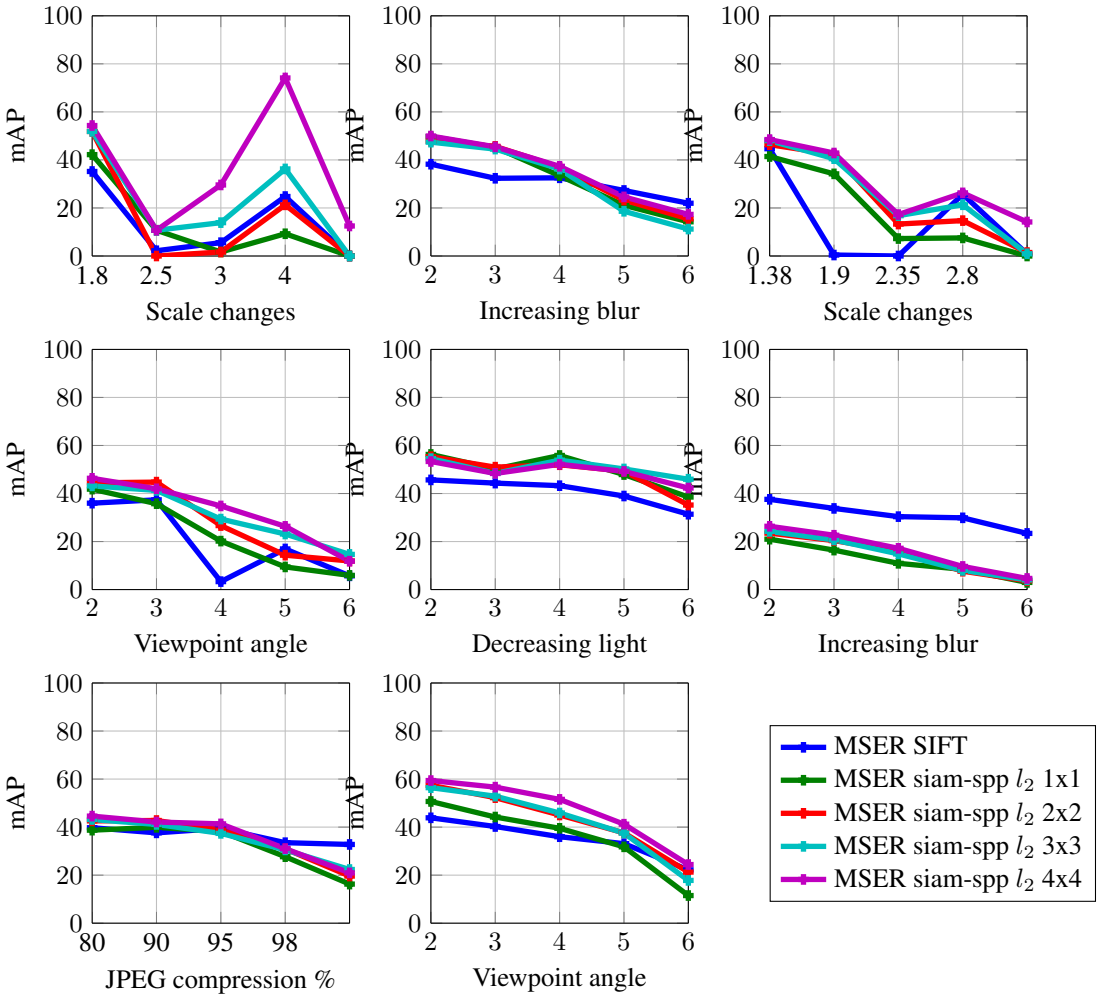


Figure 3.22: Evaluation plots of SPP-based network on different datasets when using SPP layers with different spatial sizes.

We also presented NCC-networks, improving the results even further, and proving the importance of normalization in patch-matching networks. To show the generality of presented NCC-networks we compared them with convolutional on ImageNet image classification task and showed a solid improvement. In general, as our other architectures NCC-networks can be combined among each other and the follow-up works in triplet training, leading to better convolutional descriptors in general.

Last, we should note that simply the use of a larger training set can potentially benefit and improve the overall performance of our approach even further, as the training set that was used in the present experiments can actually be considered rather small by today's standards.

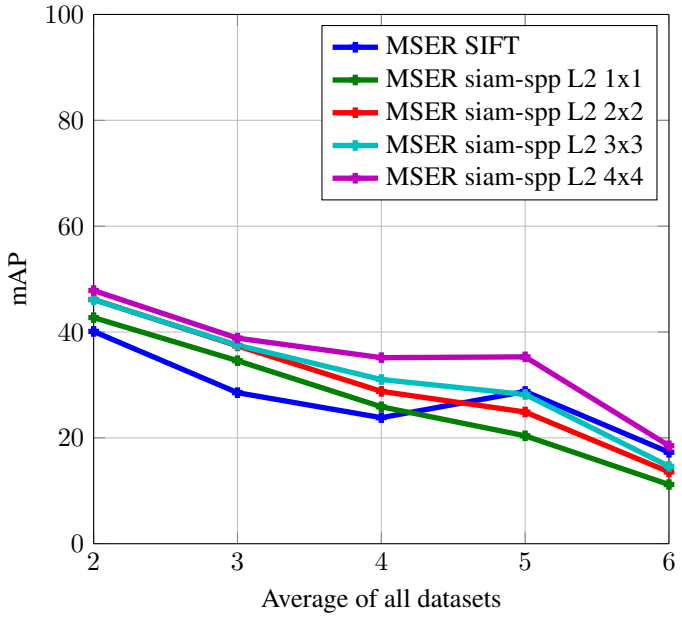


Figure 3.23: Overall performance when using SPP layers with different spatial sizes. We show average of all datasets of Fig. 3.22.

Chapter 4

Multipath neural network for object detection

Proposed in 2015, COCO object detection dataset presented several new challenges for object detection over older very popular VOC datasets. In particular, it contains objects at a broad range of scales, less prototypical images, and requires more precise localization. To address these challenges, we test three modifications to the standard Fast R-CNN object detector: (1) skip connections that give the detector access to features at multiple network layers, (2) a foveal structure to exploit object context at multiple object resolutions, and (3) an integral loss function and corresponding network adjustment that improve localization. The result of these modifications is that information can flow along multiple paths in our network, including through features from multiple network layers and from multiple object views. We refer to our modified classifier as a 'MultiPath' network. We couple our MultiPath network with DeepMask object proposals, which are well suited for localization and small objects, and adapt our pipeline to predict segmentation masks in addition to bounding boxes. The combined system improves results over the baseline Fast R-CNN detector with Selective Search by 66% overall and by $4\times$ on small objects. It placed second in both the COCO 2015 detection and segmentation challenges.

This paper is based on A MultiPath Network for Object Detection Zagoruyko *et al.* (2016).

4.1 Introduction

Object classification Krizhevsky *et al.* (2012b); Simonyan and Zisserman (2015); Szegedy *et al.* (2015) and object detection Sermanet *et al.* (2014); Szegedy *et al.* (2014); Girshick (2015) have rapidly progressed with advancements in convolutional neural networks (CNNs) LeCun *et al.* (1998) and the advent of large visual recognition datasets Everingham *et al.* (2010); Deng *et al.* (2009); Lin *et al.* (2015). Modern object detectors predominantly follow the paradigm established by Girshick et al. in their seminal work on Region CNNs Girshick *et al.* (2014): first an object proposal algorithm Hosang *et al.* (2015) generates candidate regions that may contain objects, second, a CNN classifies each proposal region. Most recent detectors follow this paradigm Gidaris and Komodakis (2015); Girshick (2015); Ren *et al.* (2015) and they have achieved rapid and impressive improvements in detection performance.

Except for concurrent work (e.g. Bell *et al.* (2016); He *et al.* (2016a); Dai *et al.* (2016)), most previous object detection work has focused on the PASCAL Everingham *et al.* (2010) and ImageNet Deng *et al.* (2009) detection datasets. Recently, the COCO dataset Lin *et al.* (2015) was introduced to push object detection to more challenging settings. The dataset contains 300,000 images of fully segmented object instance in 80 categories, with an average of 7 object instances per image. COCO introduces a number of new challenges compared to previous object detection datasets: (1) it contains objects at a broad range of scales, including a high percentage of small objects, (2) objects are less iconic, often in non-standard configurations and amid clutter or heavy occlusion, and (3) the evaluation metric encourages more accurate object localization.

In this paper, we revisit recent improvements in object detection by performing extensive experiments on the COCO dataset. In particular, we begin with the Fast R-CNN object detector Girshick (2015), and test a number of intuitive modifications to explicitly address the unique challenges of this dataset, including small object detection, detection of objects in context, and improved localization. Our goal is to adapt the highly successful Fast R-CNN object detector to perform better in these settings, and we use COCO to drive our experiments.

Inspired by recent advances in object detection, we implement three network modifications: (1) a multi-stage feature aggregator that implements skip connections in intermediate network layers to more accurately detect objects at multiple scales, (2) a foveal structure in the classifier network that helps improve localization by looking at multiple image contexts, and (3) a novel loss function and corresponding network adjustment that optimize an integral of localization overlaps and encourage higher-precision localization. These three modifications allow information to flow along multiple paths in our network,

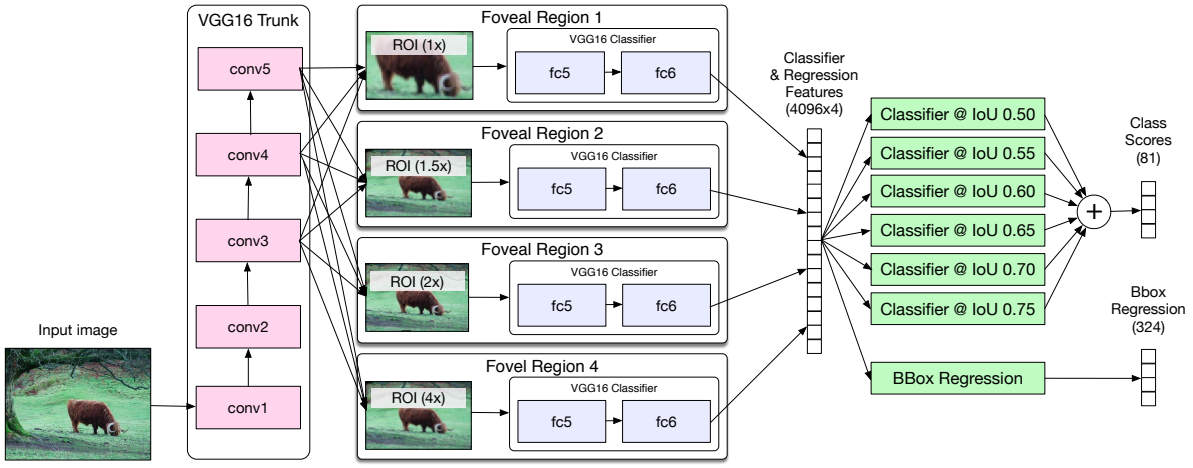


Figure 4.1: Proposed MultiPath architecture. The COCO dataset Lin *et al.* (2015) contains objects at multiple scales, in context and among clutter, and under frequent occlusion. Moreover, the COCO evaluation metric rewards high quality localization. To addresses these challenges, we propose the MultiPath network pictured above, which contains three key modifications: skip connections, foveal regions, and and an integral loss function. Together these modifications allow information to flow along multiple paths through the network, enabling the classifier to operate at multiple scales, utilize context effectively, and perform more precise object localization. Our MultiPath network, coupled with DeepMask object proposals Pinheiro *et al.* (2015, 2016), achieves major gains on COCO detection.

including through features from multiple network layers and from multiple object views, see Figure 4.1. We therefore refer to our approach as a 'MultiPath' network.

We train our MultiPath detector using the recently proposed DeepMask object proposals Pinheiro *et al.* (2015, 2016), which, like our model, are well adapted to the COCO dataset. Our combined system, using DeepMask proposals and our MultiPath classifier, achieves a detection score of 33.5 average precision (AP) for detection with an ensemble of 6 models. Compared to the baseline Fast R-CNN detector Girshick (2015) with Selective Search proposals Uijlings *et al.* (2013), which achieves an AP of 19.3, this represents a 66% improvement in performance. Moreover, for small objects we improve AP by nearly 4×. We also adopt our system to generate segmentation masks, and achieve an AP of 25.1 on the segmentation task.

Our system placed second in the 2015 COCO Detection Challenge in both the bounding box and segmentation tracks. Only the deeper ResNet classifier He *et al.* (2016a) outperformed our approach. Potentially, ResNet could be used as the feature extractor in our MultiPath network.

4.2 Related Work

Object detection is a fundamental and heavily-researched task in computer vision. Until recently, the sliding window paradigm was dominant Viola and Jones (2004); Dollár *et al.* (2014), especially for face and pedestrian detection. Deformable part models Felzenszwalb *et al.* (2010) followed this framework but allowed for more object variability and thus found success across general object categories; likewise, Sermanet et al. Sermanet *et al.* (2014, 2013) showcased the use of CNNs for general object detection in a sliding window fashion. More recent detectors follow the region-proposal paradigm established by Girshick et al. Girshick *et al.* (2014) in which a CNN is used to classify regions generated by an object proposal algorithm Hosang *et al.* (2015). Many recent detectors follow this setup Gidaris and Komodakis (2015); Szegedy *et al.* (2014); He *et al.* (2014); Girshick *et al.* (2014); Girshick (2015); Ren *et al.* (2015), including our own work, which uses the Fast R-CNN detector as its staring point Girshick (2015). We next discuss in more detail specific innovations in this paradigm and how they relate to our approach.

Context: Context is known to play an important role in visual recognition Torralba (2003). Numerous ideas for exploiting context in CNNs have been proposed. Sermanet et al. Sermanet *et al.* (2013) used two contextual regions centered on each object for pedestrian detection. In Szegedy *et al.* (2014), in addition to region specific features, features from the entire image are used to improve region classification. He et al. He *et al.* (2014) implement context in a more implicit way by aggregating CNN features prior to classification using different sized pooling regions. More recently, Gidaris and Komodakis (2015) proposed to use ten contextual regions around each object with different crops. Our approach is most related to Gidaris and Komodakis (2015), however, we use just four contextual regions organized in a foveal structure and importantly our classifier is trained jointly end-to-end.

Skip connections: Sermanet et al. Sermanet *et al.* (2013) proposed to use a 'multi-stage' classifier that used features at many convolutional layers for pedestrian detection, showing improved results. Such 'skip' architectures have recently become popular for semantic segmentation Long *et al.* (2015); Hariharan *et al.* (2015). Concurrently with our work, Bell et al. Bell *et al.* (2016) proposed to revisit skip connections for general object detection. Our own implementation of skip connections closely resembles Bell *et al.* (2016).

Object Proposals: When originally introduced, object proposals were based on low-level grouping cues, edges, and superpixels Alexe *et al.* (2012); Uijlings *et al.* (2013); Zitnick and Dollár (2014); Arbeláez *et al.* (2014); Hosang *et al.* (2015). More recently, large gains in proposal quality have been achieved

through use of CNNs Szegedy *et al.* (2014); Ren *et al.* (2015); Pinheiro *et al.* (2015, 2016). In this work we use DeepMask segmentation proposals Pinheiro *et al.* (2015). Specifically, we used an early version of the improved variant of DeepMask described in Pinheiro *et al.* (2016) that includes top-down refinement but is based on the VGG-A architecture Simonyan and Zisserman (2015), not the later ResNet architecture presented in He *et al.* (2016a). Overall, we obtain substantial improvements in detection accuracy on COCO by using DeepMask in place of the Selective Search Uijlings *et al.* (2013) proposals used in the original work on Fast R-CNN Girshick (2015).

Classifier: The CNN used for classification forms an integral part of the detection pipeline and is key in determining final detector accuracy. The field has witnessed rapid progress in CNNs in recent years. The introduction of AlexNet Krizhevsky *et al.* (2012b) reinvigorated the use of deep learning for visual recognition. The much deeper VGG Simonyan and Zisserman (2015) and GoogleNet Szegedy *et al.* (2015) models further pushed accuracy. In our work we use variants of the VGG network Simonyan and Zisserman (2015), specifically VGG-A for DeepMask and VGG-D for our MultiPath network. In concurrent work, He at al. He *et al.* (2016a) introduced the even deeper Residual Networks (ResNet) that have greatly improved the state of the art and have also proven effective for object detection. We expect that integration of ResNet into our system could further boost accuracy.

4.3 Methods

A high-level overview of our detection model is shown in Figure 4.1. Our system is based on the Fast R-CNN framework Girshick (2015). As in Fast R-CNN, the VGG-D network Simonyan and Zisserman (2015) (pretrained on ImageNet Deng *et al.* (2009)) is applied to each input image and RoI-pooling is used to extract features for each object proposal. Using these features, the final classifier outputs a score for each class (plus the background) and predicts a more precise object localization via bounding box regression. We refer readers to Girshick (2015) for details.

We propose the following modifications to this basic setup. First, instead of a single classifier head, our model has four heads that observe different-sized context regions around the bounding box in a 'foveal' structure. Second, each of these heads combines features from the conv3, conv4, and conv5 layers. Finally, the outputs of the four classifiers are concatenated and used to compute a score based on our proposed integral loss. Similar to Fast R-CNN, the network also performs bounding box regression using these same features.

As information can flow through several parallel pathways of our network we name it a MultiPath CNN. We describe details of each modification next.

4.3.1 Foveal Structure

Fast R-CNN performs RoI-pooling on the object proposal bounding box without explicitly utilizing surrounding information. However, as discussed, context is known to play an important role in object recognition Torralba (2003). We also observed that given only cropped object proposals, identification of small objects is difficult even for humans without context.

To integrate context into our model, we looked at the promising results from the 'multiregion' model Gidaris and Komodakis (2015) for inspiration. The multiregion model achieves improved localization results by focusing on 10 separate crops of an object with varying context. We hypothesized that this mainly improves localization from observing the object at multiple scales with increasing context, rather than by focusing on different parts of the object.

Therefore, to incorporate context, we add four region crops to our model with 'foveal' fields of view of $1\times$, $1.5\times$, $2\times$ and $4\times$ of the original proposal box all centered on the object proposal. In each case we use RoI-pooling to generate features maps of the same spatial dimensions given each differently-sized foveal region. The downstream processing shares an identical structure for each region (but with separate parameters), and the output features from the four foveal classifiers are concatenated into a single long vector. This feature vector is used for both classification and bounding box regression. See Figure 4.1 for details.

Our foveal model can be interpreted as a simplified version of the multiregion model that only uses four regions instead of the ten in Gidaris and Komodakis (2015). With the reduced number of heads, we can train the network end-to-end rather than each head separately as in Gidaris and Komodakis (2015).

4.3.2 Skip Connections

Fast R-CNN performs RoI-pooling after the VGG-D conv5 layer. At this layer, features have been downsampled by a factor of 16. However, 40% of COCO objects have area less than 32×32 pixels and 20% less than 16×16 pixels, so these objects will have been downsampled to 2×2 or 1×1 at this stage, respectively. RoI-pooling will upsample them to 7×7, but most spatial information will have been lost due to the $16\times$ downsampling of the features.

Effective localization of small objects requires higher-resolution features from earlier layers Sermanet *et al.* (2013); Long *et al.* (2015); Hariharan *et al.* (2015); Bell *et al.* (2016); Pinheiro *et al.* (2016). Therefore, we concatenate the RoI-pooled normalized features from conv3, conv4, and conv5 layers in the same manner as described in Bell *et al.* (2016) and provide this as input to each foveal classifier, as illustrated in Figure 4.1. A 1×1 convolution is used to reduce the dimension of the concatenated features to the classifier input dimension. The largest foveal features will not need as fine-grained features, so as an optimization, we sparsify these connections slightly. Specifically, we only connect conv3 to the $1\times$ classifier head and conv4 to the $1\times$, $1.5\times$, and $2\times$ heads. Overall, these skip connections give the classifier access to information from features at multiple resolutions.

4.3.3 Integral Loss

In PASCAL Everingham *et al.* (2010) and ImageNet Deng *et al.* (2009), the scoring metric only considers whether the detection bounding box has intersection over union (IoU) overlap greater than 50 with the ground truth. On the other hand, the COCO evaluation metric Lin *et al.* (2015) averages AP across IoU thresholds between 50 and 95, awarding a higher AP for higher-overlap bounding boxes[1]. This incentivizes better object localization. Optimizing AP^{50} has resulted in models that perform basic object localization well but often fail to return tight bounding boxes around objects.

For training, Fast R-CNN uses an IoU threshold of 50. We observed that changing this foreground/background threshold u during training improves AP^u during testing, but can decrease AP at other IoU thresholds. To target the integral AP, we propose a loss function that encourages a classifier to perform well at multiple IoU thresholds.

The original loss L used in Fast R-CNN Girshick (2015) is given by:

$$L(p, k^*, t, t^*) = L_{\text{cls}}(p, k^*) + \lambda[k^* \geq 1]L_{\text{loc}}(t, t^*), \tag{4.1}$$

for predicted class probabilities p, true class k^*, predicted bounding box t, and true bounding box t^*. The first term $L_{\text{cls}}(p, k) = -\log p_{k^*}$ is the classification log loss for true class k^*. The second term, $L_{\text{loc}}(t, t^*)$, encourages the class-specific bounding box prediction to be as accurate as possible. The combined loss is computed for every object proposal. If the proposal overlaps a ground truth box with

[1] Going forward, we use the notation introduced by the COCO dataset Lin *et al.* (2015). Specifically, we use AP to denote AP averaged across IoU values from 50 to 95, and AP^u to denote AP at IoU threshold u (e.g., the PASCAL metric is denoted by AP^{50}). Note also that we use the convention that IoU ranges from 0 to 100.

IoU greater than 50, the true class k^* is given by the class of the ground truth box, otherwise $k^* = 0$ and the second term of the loss is ignored.

Observe that in the original R-CNN loss, the classification loss L_{cls} does not prefer object proposals with high IoU: all proposals with IoU greater than 50 are treated equally. Ideally, proposals with higher overlap to the ground truth should be scored more highly. We thus propose to modify L_{cls} to explicitly measure integral loss over all IoU thresholds u:

$$\int_{50}^{100} L_{\text{cls}}(p, k_u^*)du, \tag{4.2}$$

where k_u^* is the true class at overlap threshold u. We approximate this integral as a sum with $du = 5$ and modify our network to output multiple corresponding predictions p_u. Specifically, our modified loss can be written as:

$$L(p, k^*, t, t^*) = \frac{1}{n}\sum_u \Big[L_{\text{cls}}(p_u, k_u^*) + \lambda[k_u^* \geq 1]L_{\text{loc}}(t, t^*)\Big]. \tag{4.3}$$

We use $n = 6$ thresholds $u \in \{50, 55, \ldots, 75\}$. Note that in this formulation each object proposal actually has n ground truth labels k_u^*, one label per threshold u. In our model, each term p_u is predicted by a separate head, see Figure 4.1. Specifically, for each u, we train a separate linear classifier (using shared features) to predict the true class k_u^* of a proposal (where the ground truth label is defined using threshold u). At inference time, the output softmax probabilities p_u of each of the n classifiers are averaged to compute the final class probabilities p. The modified loss function and updated network encourages object proposals with higher overlap to the ground truth to be scored more highly.

During training, each head has progressively fewer total positive training samples as there are fewer proposals overlapping the ground truth as u is increased. To keep the ratio of sampled positive and negative examples constant for each head, each minibatch is constructed to train a single head in turn. We restrict the heads to the range $u \leq 75$, otherwise the proposals contain too few total positive samples for training. Finally, note that for bounding box regression, our network is unchanged and predicts only a single bounding box output t.

4.4 Experiments

In this section we perform a detailed experimental analysis of our MultiPath network. For all following experiments, Fast R-CNN Girshick (2015) serves as our baseline detector (with VGG-D Simonyan and Zisserman (2015) features pre-trained on ImageNet Deng *et al.* (2009)). We use DeepMask object

proposals Pinheiro *et al.* (2015, 2016) and focus exclusively on the recent COCO dataset Lin *et al.* (2015) which presents novel challenges for detection.

We begin by describing the training and testing setup in §4.4.1. Next, in §4.4.2 we study the impact of each of our three core network modifications, including skip connections, foveal regions, and the integral loss. We analyze the gain from DeepMask proposals in §4.4.3 and compare with the state of the art in §4.5. Finally, in the section 4.5 we analyze a number of key parameters and also additional modifications that by and large did *not* improve accuracy.

Our system is written using the Torch-7 framework.

4.4.1 Training and Testing Setup

For all experiments in this section we report both the overall AP (averaged over multiple IoU thresholds) and AP^{50}. All our models are trained on the 80K images in COCO 2014 train set and tested on the first 5K images from the val set. We find that testing on these 5K images correlates well with the full 40K val set and 20K test-dev set, making these 5K images a good proxy for model validation without the need to test over the full val or test-dev sets.

Training is performed for 200K iterations with 4 images per batch and 64 object proposals per image. We use an initial learning rate of 10^{-3} and reduce it to 10^{-4} after 160K iterations. Training the full model takes ∼3 days on 4 NVIDIA Titan X GPUs. Unless noted, in testing we use a non maximal suppression threshold of 30, 1000 proposals per image, an image scale of 800 pixels, and no weight decay (we analyze all settings in section 4.5).

Both data and model parallelism are used in training Krizhevsky (2014). First, 4 images are propagated through the VGG-D network trunk, in parallel with 1 image per GPU. The features are then concatenated into one minibatch and subsequently used by each of the 4 foveal regions. Each foveal region resides in a separate GPU. Note that the prevalence of 4 GPU machines helped motivate our choice of using 4 foveal regions due to ease of parallelization.

Our network requires 150ms to compute the features and 350ms to evaluate the foveal regions, for a total of about 500ms per COCO image. We time with a scale of 800px and 400 proposals (see section 4.5 and Figure 4.3). Fast R-CNN with these settings is about 2× faster.

integral loss	foveal	skip	AP^{50}	AP
			43.4	25.2
✓			42.2	25.6
	✓		45.2	25.8
✓	✓		44.4	26.9
	✓	✓	**46.4**	27.0
✓	✓	✓	44.8	**27.9**

integral loss	context	#regions	AP^{50}	AP
	none	1	43.4	25.2
	multiregion	10	44.0	25.5
	foveal	4	**45.2**	**25.8**
✓	none	1	42.2	25.6
✓	multiregion	10	43.1	26.3
✓	foveal	4	**44.4**	**26.9**

Table 4.1: **Left:** Model improvements of our MultiPath network. Results are shown for various combinations of modifications enabled. Each contributes roughly equally to final accuracy, and in total AP increases 2.7 points to 27.9. **Right:** Our 4-region foveal setup versus the 10 regions used in multiregion Gidaris and Komodakis (2015). Surprisingly, our approach outperforms Gidaris and Komodakis (2015) despite using fewer regions. See text for details.

4.4.2 MultiPath Network Analysis

Our implementation of Fast R-CNN Girshick (2015) with DeepMask object proposals Pinheiro *et al.* (2015) achieves an overall AP of 25.2 and an AP^{50} of 43.4. This is already a large improvement over the original Fast R-CNN results that used Selective Search proposals Uijlings *et al.* (2013), we will return to this shortly.

A breakdown of how each of our three core network modifications affects AP and AP^{50} over our strong baseline is shown in Table 4.1, left. Results are shown for each combination of modifications enabled including skip connections, foveal regions, and the integral loss (except skip connections were implemented only for foveal regions). Altogether, AP increases 2.7 points to 27.9, with each modification contributing ~1 point to final performance. AP^{50} improves 1.4 points to 44.8; however, not surprisingly, the best AP^{50} of 46.4 is achieved without the integral loss. We carefully analyze the foveal structure and integral loss next.

Foveal structure: A breakdown of the gains from using foveal regions is shown in Table 4.1, right, both with and without the integral loss but without skip connections. Gains from foveal regions are amplified when using the integral loss, resulting in an AP improvement of 1.3 points. We also compare our foveal approach to the multiregion network Gidaris and Komodakis (2015) which used 10 regions (for a fair comparison, we re-implement it in our setup). Surprisingly, it performs slightly *worse* than our foveal setup despite having more regions. This may be due to the higher number of parameters or it's possible that this requires more iterations to converge.

Integral Loss: Figure 4.2, left, shows AP at various IoU thresholds for models trained with different IoU cutoffs u as well as our integral loss. Each standard model tends to perform best at the IoU for which it was trained. Integral loss improves overall AP by ~1 over the $u = 50$ model, and does so while

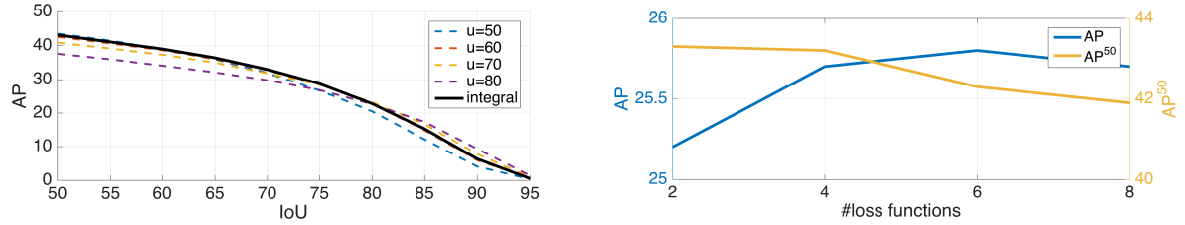


Figure 4.2: **Left**: Each standard model performs best at the threshold used for training while using the integral loss yields good results at all settings. **Right**: Integral loss achieves best AP with 6 heads.

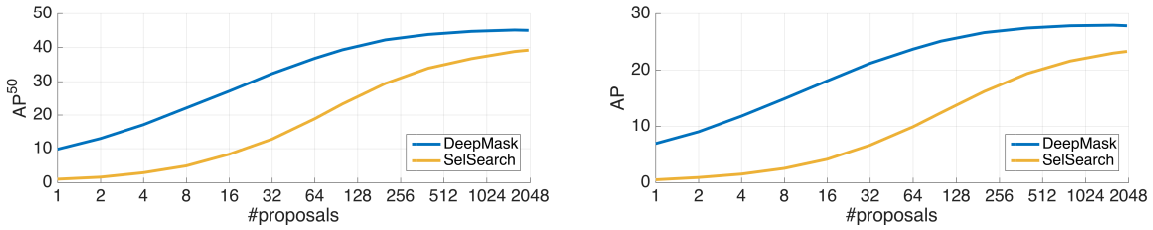


Figure 4.3: AP^{50} and AP versus number and type of proposals. Accuracy saturates using 400 DeepMask proposals per image and using ~50 DeepMask proposals matches 2000 Selective Search proposals.

maintaining a slightly higher AP^{50} than simply increasing u (e.g. our AP^{50} is 0.6 points higher than the $u = 60$ model). Figure 4.2, right, shows AP and AP^{50} for varying number of heads. Using 6 heads ($u \leq 75$) achieves the highest AP. For the experiments in Figure 4.2 we trained for 280K iterations as we found the integral loss requires somewhat longer to converge (we used 200K iterations for all other ablations studies).

4.4.3 DeepMask Proposals

Object proposals play a central role in determining detector accuracy. The original implementation of Fast R-CNN with Selective Search proposals Uijlings *et al.* (2013) has an AP of 19.3. Our MultiPath network improves this to 22.8 AP using these same proposals. Switching to DeepMask proposals Pinheiro *et al.* (2015, 2016) increases accuracy by a further very substantial 5.1 points to 27.9 AP.

Figure 4.3 shows AP^{50} and AP for varying number and type of proposals. Not only is accuracy substantially higher using DeepMask, fewer proposals are necessary to achieve top performance. Our results saturate with around 400 DeepMask proposals per image and using just 50 DeepMask proposals matches accuracy with 2000 Selective Search proposals.

Interestingly, our setup substantially reduces the benefits provided by bounding box regression. With the original Fast R-CNN and Selective Search proposals, box regression increases AP by 3.5 points, but

	AP^{50}			AP		
	base	+bb	Δ	base	+bb	Δ
SS + Fast R-CNN	38.2	39.8	+1.6	18.1	21.6	+3.5
SS + MultiPath	38.0	38.5	+0.5	20.9	22.8	+1.9
DM + Fast R-CNN	42.5	43.4	+0.9	23.5	25.2	+1.7
DM + MultiPath	44.5	44.8	+0.3	26.8	27.9	+1.1

	AP^{50}	Δ	AP	Δ
baseline	44.8		27.9	
+ trainval	47.5	+2.7	30.2	+2.3
+ hflip	48.3	+0.8	30.8	+0.6
+ FMP	49.6	+1.3	31.5	+0.7
+ ensembling	51.9	+2.3	33.2	+1.7

Table 4.2: **Left:** Bounding box regression is key when using Selective Search (SS) proposals and the Fast R-CNN classifier (our implementation). However, with DeepMask (DM) proposals and our MultiPath network, box regression increases AP by only 1.1 points (and AP^{50} by 0.3) as our pipeline already outputs well-localized detections. **Right:** Final enhancements to our model. Use of additional training data, horizontal flip at inference, fractional max pooling (FMP), and ensembling gave a major cumulative boost. These are common approaches for maximizing accuracy, see section. 4.5 for details.

Figure 4.4: Selected detection results on COCO. Only high-scoring detections are shown. While there are missed objects and false positives, many of the detections and segmentations are quite good.

	AP	AP^{50}	AP^{75}	AP^{S}	AP^{M}	AP^{L}	AR^{1}	AR^{10}	AR^{100}	AR^{S}	AR^{M}	AR^{L}
ResNet He *et al.* (2016a)	27.9	51.2	27.6	8.6	30.2	45.3	25.4	37.1	38.0	16.6	43.3	57.8
MultiPath	25.0	45.4	24.5	7.2	28.8	39.0	23.8	36.6	38.5	17.0	46.7	53.5
ResNet He *et al.* (2016a)	37.1	58.8	39.8	17.3	41.5	52.5	31.9	47.5	48.9	26.7	55.2	67.9
MultiPath	33.2	51.9	36.3	13.6	37.2	47.8	29.9	46.0	48.3	23.4	56.0	66.4
ION Bell *et al.* (2016)	30.7	52.9	31.7	11.8	32.8	44.8	27.7	42.8	45.4	23.0	50.1	63.0
Fast R-CNN* Girshick (2015)	19.3	39.3	19.9	3.5	18.8	34.6	21.4	29.5	29.8	7.7	32.2	50.2
Faster R-CNN* Ren *et al.* (2015)	21.9	42.7	—	—	—	—	—	—	—	—	—	—

Table 4.3: **Top**: COCO test-standard segmentation results. **Bottom**: COCO test-standard bounding box results (top methods only). Leaderboard snapshot from 01/01/2016. *Note: Fast R-CNN and Faster R-CNN results are on test-dev as reported in Ren *et al.* (2015), but results between splits tend to be quite similar.

with our MultiPath model and DeepMask proposals, box regression only increases AP by 1.1 points. See Table 4.2, left, for details.

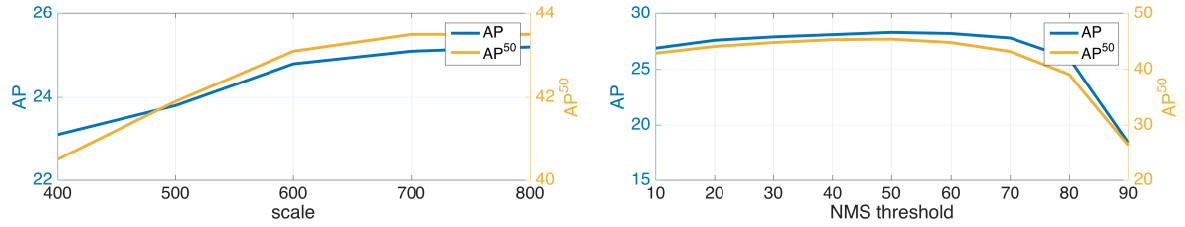


Figure 4.5: Effect of scale (left) and NMS threshold (right) on detection performance

4.5 COCO 2015 Results

To maximize accuracy prior to submitting to the COCO leaderboard, we added validation data to training, employed horizontal flip and fractional max pooling Graham (2014) at inference, and ensembled 6 models. Together, these four enhancements boosted AP from 27.9 to 33.2 on the held-out validation images, see Table 4.2, right. More details are given in the section 4.5. Finally, to obtain segmentation results, we simply fed the bounding box regression outputs back to the DeepMask segmentation system. Note that as discussed in §4.4.3, box regression only improved accuracy slightly. In principle, we could have used the original DeepMask segmentation proposals without box regression; however, we did not test this variant.

We submitted our results the COCO 2015 Object Detection Challenge. Our system placed second in both the bounding box and segmentation tracks. Table 4.3 compares our results to the top COCO 2015 challenge systems and additional baselines. Only the deeper ResNet classifier He *et al.* (2016a) outperformed our approach (and potentially ResNet could be integrated as the feature extractor in our MultiPath network, leading to further gains). Compared to the baseline Fast R-CNN, our system showed the largest gains on small objects and localization, improving AP on small objects by $4\times$ and AP^{75} by 82%.

Figure 4.4 and Figure 4.6 show selected detection results from our system. Figure 4.7 shows a breakdown of errors of our system. Most of the overall error comes from false positives and negatives, with little inter-class classification error. Despite our improvements on small objects, small object detection remains quite challenging.

Additional Analysis

In this section we describe our additional enhancements reported in Table 4.2 and analyze a number of key parameters. We also report additional modifications that did *not* improve accuracy; we hope that sharing our *negative results* will prove beneficial to the community.

train+val: Adding validation data to training (minus the 5K held-out images from the validation set we use for testing) improved accuracy by 2.3 points AP, see Table 4.2. We trained for 280K iterations in this case. We note that the DeepMask proposals were only trained using the train set, so retraining these on train+val could further improve results.

hflip: Fast R-CNN is not invariant to horizontal image flip (hflip) even though it is trained with hflip data augmentation. Thus, we average the softmax scores from the original and flipped images and also average the box regression outputs (directly, not in log space). AP improves by 0.6 points, see Table 4.2.

FMP: Inspired by Fractional Max Pooling Graham (2014), we perform multiple RoI-pooling operations with perturbed pooling parameters and average the softmax outputs (note that the network trunk is computed only once). Specifically, we perform two ROI-poolings: the first follows He *et al.* (2014) and uses the floor and ceil operations for determining the RoI region, the second uses the round operation. As shown in Table 4.2, FMP improves AP 0.7 points.

Ensembling: Finally, we trained an ensemble of 6 similar models. Each model was initialized with the same ImageNet pre-trained model, only the order of COCO training images changed. This ensemble boosted AP 1.7 points to 33.2, see Table 4.2.

Scale: Figure 4.5, left, shows accuracy as a function of image scale (minimum image dimension in pixels with maximum fixed to 1000px). Increasing scale improves accuracy up to ~800px, but at increasing computation time. We set the scale to 800px which improves AP by 0.5 points over the 600px scale used by Girshick (2015) for PASCAL.

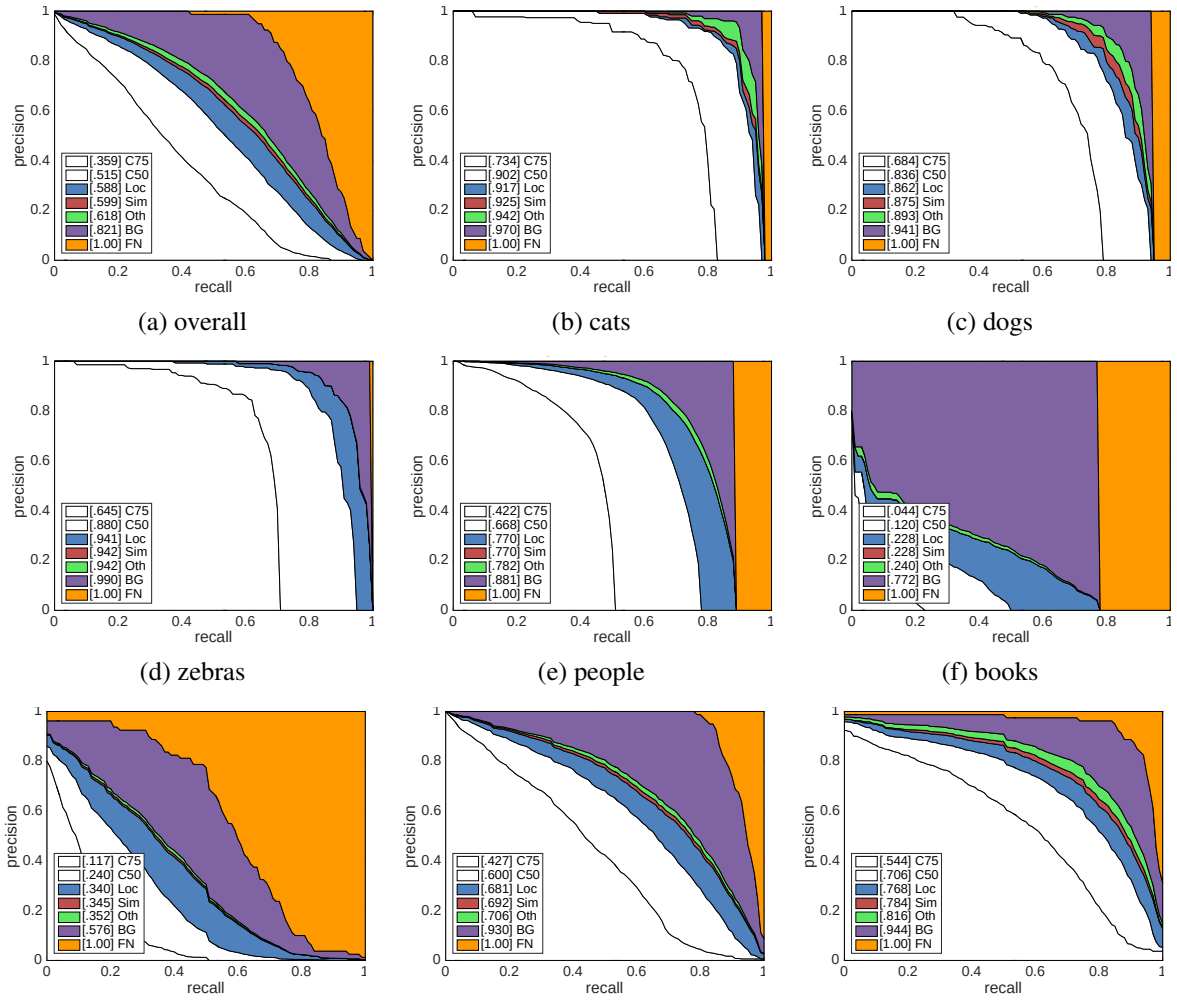
NMS threshold: Figure 4.5, right, shows accuracy as a function of the NMS threshold. Fast R-CNN Girshick (2015) used a threshold of 30. For our model, an NMS threshold of 50 performs best, improving AP by 0.4 points, possibly due to the higher object density in COCO.

Dropout & Weight Decay: Dropout helped regularize training and we keep the same dropout value of 0.5 that was used for training VGG-D. On the other hand, setting weight decay to 0 for fine-tuning improved results by 1.1 AP^{50} and 0.5 AP. Note that Bell *et al.* (2016) used weight decay but not dropout, so perhaps it is sufficient to have just one form of regularization.

Iterative Localization: Bounding box voting with iterative localization as proposed in Gidaris and Komodakis (2015) did *not* substantially improve the AP of our model, again probably due to the higher quality of DeepMask proposals and the improved localization ability of our MultiPath network.

ImageNet Data Augmentation: As there are some under-represented classes in COCO with few annotations, we tried to augment the training set with ImageNet 2012 detection training data. Surprisingly, this only improved performance on the most underrepresented class: hair dryer; for all other classes, accuracy remained unchanged or suffered.

Figure 4.6: Selected detection results on COCO. Only high-scoring detections are shown. While there are missed objects and false positives, many of the detections and segmentations are quite good.



(a) overall (b) cats (c) dogs

(d) zebras (e) people (f) books

(g) small (h) medium (i) large

Figure 4.7: Detailed analysis of detector performance on unseen COCO validation images at select settings (plots in style of Hoiem *et al.* (2012) generated by COCO API code). **(a)** Removing localization errors would lead to an AP^{10} of 58.8 on COCO ('Loc'). Removing similar and other class confusion ('Sim' and 'Oth') would only lead to slight improvements in accuracy. The remaining errors are all based on background confusions ('BG') and false negatives ('FN'). **(b,c)** Our detector performs similarly on cats and dogs, achieving high overall accuracy with some class and background confusions but few missed detections. **(d)** Zebras are quite distinct, however, localization of overlapping zebras can be difficult due to their striped patterns. **(e)** People are the dominant category on COCO and have average difficulty. **(f)** Books are an incredibly difficult category due to their small size and highly inconsistent annotation in COCO. **(g,h,i)** Accuracy broken down by scale; not unexpectedly, small objects ($area < 32^2$) are quite difficult, while accuracy on large objects ($area > 96^2$) is much higher. While there is a practical limit to the performance on small objects which are often ambiguous or poorly-labeled, there is still substantial opportunity for improvement. We expect better proposals, more accurate filtering of false positives, and stronger reasoning about context can all improve small object detection.

4.6 Conclusions

In this chapter, we proposed three modifications to Fast R-CNN: (1) skip connections to give the network access to multi-scale features, (2) foveal regions to provide context, and (3) the integral loss to improve localization. We coupled our resulting MultiPath classifier with DeepMask proposals and achieved a 66% improvement over the baseline Fast R-CNN with Selective Search.

Chapter 5

Residual weight parameterizations in deep neural networks

In this chapter we address the problem of understanding deep networks with residual weight parameterizations, which were shown to be able to scale up to thousands of layers and still have improving performance. However, each fraction of a percent of improved accuracy costs nearly doubling the number of layers, and so training very deep residual networks has a problem of diminishing feature reuse, which makes these networks very slow to train. To tackle these problems, in the first part of this chapter we conduct a detailed experimental study on the architecture of ResNet blocks, based on which we propose a novel architecture where we decrease depth and increase width of residual networks. We call the resulting network structures wide residual networks (WRNs) and show that these are far superior over their commonly used thin and very deep counterparts.

We also observe that the initial motivation behind ResNet - training deeper networks - does not actually hold true, and the benefits come from increased capacity, rather than from depth. Based on this, in the second part we explore alternative definitions of ResNet, and propose an implicit skip-connection via weight parameterization as a sum of weight and Dirac delta function. This parameterization has a minor computational cost at training time and no cost at all at inference, as both Dirac parameterization and batch normalization can be folded into convolutional filters, so that network becomes a simple chain of convolution-ReLU pairs.

The chapter is based on Wide Residual Networks Zagoruyko and Komodakis (2016b) and DiracNets: Training Very Deep Neural Networks Without Skip-Connections Zagoruyko and Komodakis (2017a).

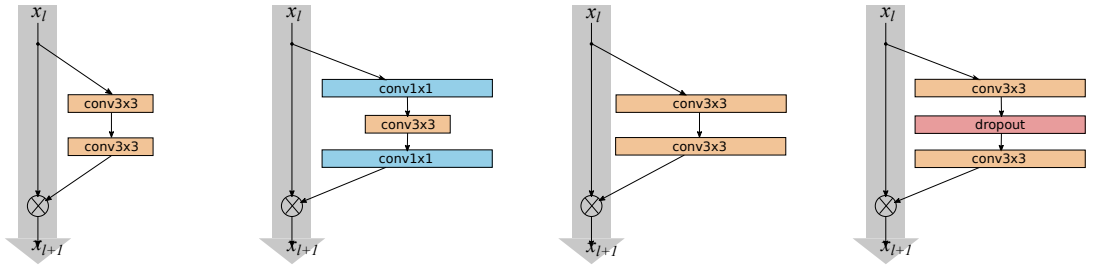
5.1 Introduction

Convolutional neural networks have seen a gradual increase of the number of layers in the last few years, starting from AlexNet Krizhevsky *et al.* (2012b), VGG Simonyan and Zisserman (2015), Inception Szegedy *et al.* (2015) to Residual He *et al.* (2016a) networks, corresponding to improvements in many image recognition tasks. The superiority of deep networks has been spotted in several works in the recent years Bianchini and Scarselli (2014); Montúfar *et al.* (2014). However, training deep neural networks has several difficulties, including exploding/vanishing gradients and degradation. Various techniques were suggested to enable training of deeper neural networks, such as well-designed initialization strategies Bengio and Glorot (2010); He *et al.* (2015), better optimizers Sutskever *et al.* (2013), skip connections Lee *et al.* (2014); Raiko *et al.* (2012), knowledge transfer Romero *et al.* (2014); Chen *et al.* (2016) and layer-wise training Schmidhuber (1992).

The latest residual networks He *et al.* (2016a), a follow-up of highway networks Srivastava *et al.* (2015), had a large success winning ImageNet and COCO 2015 competition and achieving state-of-the-art in several benchmarks, including object classification on ImageNet and CIFAR, object detection and segmentation on PASCAL VOC and MS COCO. Compared to Inception architectures they show better generalization, meaning the features can be utilized in transfer learning with better efficiency. Also, follow-up work showed that residual links speed up convergence of deep networks Szegedy *et al.* (2016). Recent follow-up work explored the order of activations in residual networks, presenting identity mappings in residual blocks He *et al.* (2016b) and improving training of very deep networks. The essential difference between residual and highway networks is that in the latter residual links are gated and weights of these gates are learned.

So, up to this point, the study of residual networks has focused mainly on the order of activations inside a ResNet block and the depth of residual networks. In this chapter we attempt to conduct an experimental study that goes beyond the above points. By doing so, our goal is to explore a much richer set of network architectures of ResNet blocks and thoroughly examine how several other different aspects besides the order of activations affect performance. As we explain below, such an exploration of architectures has led to new interesting findings with great practical importance concerning residual networks.

Width vs depth in residual networks. The problem of shallow vs deep networks has been in discussion for a long time in machine learning Larochelle *et al.* (2007); Bengio and LeCun (2007) with pointers to the circuit complexity theory literature showing that shallow circuits can require exponentially more components than deeper circuits. The authors of residual networks tried to make them as thin as possible



(a) basic (b) bottleneck (c) basic-wide (d) wide-dropout

Figure 5.1: Various residual blocks used in the chapter. Batch normalization and ReLU precede each convolution (omitted for clarity)

in favor of increasing their depth and having less parameters, and even introduced a "bottleneck" block which makes ResNet blocks even thinner.

We note, however, that the residual block with identity mapping that allows to train very deep networks is at the same time a weakness of residual networks. As gradient flows through the network there is nothing to force it to go through residual block weights and it can avoid learning anything during training, so it is possible that there is either only a few blocks that learn useful representations, or many blocks share very little information with small contribution to the final goal. This problem was formulated as diminishing feature reuse in Srivastava *et al.* (2015). The authors of Huang *et al.* (2016) tried to address this problem with the idea of randomly disabling residual blocks during training. This method can be viewed as a special case of dropout Srivastava *et al.* (2014), where each residual block has an identity scalar weight on which dropout is applied. The effectiveness of this approach proves the hypothesis above.

Motivated by the above observation, our work builds on top of He *et al.* (2016b) and tries to answer the question of how wide deep residual networks should be and address the problem of training. In this context, we show that the widening of ResNet blocks (if done properly) provides a much more effective way of improving performance of residual networks compared to increasing their depth. In particular, we present wider deep residual networks that significantly improve over He *et al.* (2016b), having *50 times* less layers and being more than 2 times faster. We call the resulting network architectures *wide residual networks*. For instance, our wide 16-layer deep network has the same accuracy as a 1000-layer thin deep network and a comparable number of parameters, although being several times faster to train. This type of experiments thus seem to indicate that the main power of deep residual networks is in residual blocks, and that the effect of depth is supplementary. We note that one can train even better wide residual networks that have twice as many parameters (and more), which suggests that to further improve performance by increasing depth of thin networks one needs to add thousands of layers in this case.

Use of dropout in ResNet blocks. Dropout was first introduced in Srivastava *et al.* (2014) and then was adopted by many successful architectures as Krizhevsky *et al.* (2012b); Simonyan and Zisserman (2015) etc. It was mostly applied on top layers that had a large number of parameters to prevent feature coadaptation and overfitting. It was then mainly substituted by batch normalization Ioffe and Szegedy (2015) which was introduced as a technique to reduce internal covariate shift in neural network activations by normalizing them to have specific distribution. It also works as a regularizer and the authors experimentally showed that a network with batch normalization achieves better accuracy than a network with dropout. In our case, as widening of residual blocks results in an increase of the number of parameters, we studied the effect of dropout to regularize training and prevent overfitting. Previously, dropout in residual networks was studied in He *et al.* (2016b) with dropout being inserted in the identity part of the block, and the authors showed negative effects of that. Instead, we argue here that dropout should be inserted between convolutional layers. Experimental results on wide residual networks show that this leads to consistent gains, yielding even new state-of-the-art results (e.g. 16-layer-deep wide residual network with dropout achieves 1.64% error on SVHN).

In summary, the contributions of this work are as follows:

- We present a detailed experimental study of residual network architectures that thoroughly examines several important aspects of ResNet block structure.
- We propose a novel *widened* architecture for ResNet blocks that allows for residual networks with significantly improved performance.
- We propose a new way of utilizing dropout within deep residual networks so as to properly regularize them and prevent overfitting during training.
- Last, we show that our proposed ResNet architectures achieve state-of-the-art results on several datasets dramatically improving accuracy and speed of residual networks.

5.2 Wide Residual Networks

Residual block with identity mapping can be represented by the following formula:

$$\boldsymbol{x}_{l+1} = \boldsymbol{x}_l + \mathcal{F}(\boldsymbol{x}_l, \boldsymbol{W}_l),$$

where $\boldsymbol{x}_{l+1}$ and $\boldsymbol{x}_l$ are input and output of the l-th unit in the network, $\mathcal{F}$ is a residual function and $\boldsymbol{W}_l$ are parameters of the block. Residual network consists of sequentially stacked residual blocks.

In He *et al.* (2016b) residual networks consisted of two type of blocks:

- *basic* - with two consecutive 3×3 convolutions with batch normalization and ReLU preceding convolution: `conv3 × 3–conv3 × 3` Fig.5.1a
- *bottleneck* - with one 3×3 convolution surrounded by dimensionality reducing and expanding 1×1 convolution layers: `conv1 × 1–conv3 × 3–conv1 × 1` Fig.5.1b

Compared to the original architecture He *et al.* (2016a) in He *et al.* (2016b) the order of batch normalization, activation and convolution in residual block was changed from `conv-BN-ReLU` to `BN-ReLU-conv`. As the latter was shown to train faster and achieve better results we don't consider the original version. Furthermore, so-called "bottleneck" blocks were initially used to make blocks less computationally expensive to increase the number of layers. As we want to study the effect of widening and "bottleneck" is used to make networks thinner we don't consider it too, focusing instead on "basic" residual architecture.

There are essentially three simple ways to increase representational power of residual blocks:

- to add more convolutional layers per block
- to widen the convolutional layers by adding more feature planes
- to increase filter sizes in convolutional layers

As small filters were shown to be very effective in several works including Simonyan and Zisserman (2015); Szegedy *et al.* (2016) we do not consider using filters larger than 3×3. Let us also introduce two factors, deepening factor l and widening factor k, where l is the number of convolutions in a block and k multiplies the number of features in convolutional layers, thus the baseline "basic" block corresponds to $l = 2$, $k = 1$. Figures 5.1a and 5.1c show schematic examples of "basic" and "basic-wide" blocks respectively.

The general structure of our residual networks is illustrated in table 5.1: it consists of an initial convolutional layer `conv1` that is followed by 3 groups (each of size N) of residual blocks `conv2`, `conv3` and `conv4`, followed by average pooling and final classification layer. The size of `conv1` is fixed in all of our experiments, while the introduced widening factor k scales the width of the residual blocks in the three groups `conv2–4` (e.g. the original "basic" architecture is equivalent to $k = 1$). We want to study the effect of representational power of residual block and, to that end, we perform and test several modifications to the "basic" architecture, which are detailed in the following subsections.

group name	output size	block type = $B(3,3)$
conv1	32×32	[3×3, 16]
conv2	32×32	$\begin{bmatrix} 3\times3, 16\times k \\ 3\times3, 16\times k \end{bmatrix} \times N$
conv3	16×16	$\begin{bmatrix} 3\times3, 32\times k \\ 3\times3, 32\times k \end{bmatrix} \times N$
conv4	8×8	$\begin{bmatrix} 3\times3, 64\times k \\ 3\times3, 64\times k \end{bmatrix} \times N$
avg-pool	1×1	$[8 \times 8]$

Table 5.1: Structure of wide residual networks. Network width is determined by factor k. Original architecture He *et al.* (2016b) is equivalent to $k = 1$. Groups of convolutions are shown in brackets where N is a number of blocks in group, downsampling performed by the first layers in groups `conv3` and `conv4`. Final classification layer is omitted for clarity. In the particular example shown, the network uses a ResNet block of type $B(3,3)$.

5.2.1 Type of convolutions in residual block

Let $B(M)$ denote residual block structure, where M is a list with the kernel sizes of the convolutional layers in a block. For example, $B(3,1)$ denotes a residual block with 3×3 and 1×1 convolutional layers (we always assume square spatial kernels). Note that, as we do not consider "bottleneck" blocks as explained earlier, the number of feature planes is always kept the same across the block. We would like to answer the question of how important each of the 3×3 convolutional layers of the "basic" residual architecture is and if they can be substituted by a less computationally expensive 1×1 layer or even a combination of 1×1 and 3×3 convolutional layers, e.g. $B(1,3)$ or $B(1,3)$. This can increase or decrease the representational power of the block. We thus experiment with the following combinations (note that the last combination, i.e., $B(3,1,1)$ is similar to effective Network-in-Network Lin *et al.* (2013) architecture):

1. $B(3,3)$ - original "basic" block
2. $B(3,1,3)$ - with one extra 1×1 layer
3. $B(1,3,1)$ - with the same dimensionality of all convolutions, "straightened" bottleneck
4. $B(1,3)$ - the network has alternating 1×1 - 3×3 convolutions everywhere
5. $B(3,1)$ - similar idea to the previous block
6. $B(3,1,1)$ - Network-in-Network style block

5.2.2 Number of convolutional layers per residual block

We also experiment with the block deepening factor l to see how it affects performance. The comparison has to be done among networks with the same number of parameters, so in this case we need to build networks with different l and d (where d denotes the total number of blocks) while ensuring that network complexity is kept roughly constant. This means, for instance, that d should decrease whenever l increases.

5.2.3 Width of residual blocks

In addition to the above modifications, we experiment with the widening factor k of a block. While the number of parameters increases linearly with l (the deepening factor) and d (the number of ResNet blocks), number of parameters and computational complexity are quadratic in k. However, it is more computationally effective to widen the layers than have thousands of small kernels as GPU is much more efficient in parallel computations on large tensors, so we are interested in an optimal d to k ratio.

One argument for wider residual networks would be that almost all architectures before residual networks, including the most successful Inception Szegedy *et al.* (2015) and VGG Simonyan and Zisserman (2015), were much wider compared to He *et al.* (2016b). For example, residual networks WRN-22-8 and WRN-16-10 (see next paragraph for explanation of this notation) are very similar in width, depth and number of parameters to VGG architectures.

We further refer to original residual networks with $k = 1$ as "thin" and to networks with $k > 1$ as "wide". In the rest of the chapter we use the following notation: WRN-n-k denotes a residual network that has a total number of convolutional layers n and a widening factor k (for example, network with 40 layers and $k = 2$ times wider than original would be denoted as WRN-40-2). Also, when applicable we append block type, e.g. WRN-40-2-$B(3, 3)$.

5.2.4 Dropout in residual blocks

As widening increases the number of parameters we would like to study ways of regularization. Residual networks already have batch normalization that provides a regularization effect, however it requires heavy data augmentation, which we would like to avoid, and it's not always possible. We add a dropout layer into each residual block between convolutions as shown in fig. 5.1d and after ReLU to perturb batch normalization in the next residual block and prevent it from overfitting. In very deep residual

networks that should help deal with diminishing feature reuse problem enforcing learning in different residual blocks.

5.2.5 Experimental results

All of our experiments are based on He *et al.* (2016b) architecture with pre-activation residual blocks and we use it as baseline. For experiments we chose well-known CIFAR-10, CIFAR-100, SVHN and ImageNet image classification datasets. CIFAR-10 and CIFAR-100 datasets Krizhevsky *et al.* (2012a) consist of 32×32 color images drawn from 10 and 100 classes split into 50,000 train and 10,000 test images. For data augmentation we do horizontal flips and take random crops from image padded by 4 pixels on each side, filling missing pixels with reflections of original image. We don't use heavy data augmentation as proposed in Graham (2014). SVHN is a dataset of Google's Street View House Numbers images and contains about 600,000 digit images, coming from a significantly harder real world problem. For experiments on SVHN we don't do any image preprocessing, except dividing images by 255 to provide them in [0,1] range as input. To speed up training we run "type of convolutions in a block" and "number of convolutions per block" experiments with $k = 2$ and reduced depth compared to He *et al.* (2016b).

Initially we followed CIFAR image preprocessing of Goodfellow *et al.* (2013) with ZCA whitening, but later found out that simple mean/std normalization was used in He *et al.* (2016b) and other ResNet related works, so we updated tables where comparison with other methods needed. We further use ZCA preprocessing, unless mentioned otherwise.

In the following we describe our findings w.r.t. the different ResNet block architectures and also analyze the performance of our proposed wide residual networks.

block type	depth	# params	time,s	CIFAR-10
$B(1,3,1)$	40	1.4M	85.8	6.06
$B(3,1)$	40	1.2M	67.5	5.78
$B(1,3)$	40	1.3M	72.2	6.42
$B(3,1,1)$	40	1.3M	82.2	5.86
$B(3,3)$	28	1.5M	67.5	5.73
$B(3,1,3)$	22	1.1M	59.9	5.78

Table 5.2: Test error (%, median over 5 runs) on CIFAR-10 of residual networks with $k = 2$ and different block types. Time column measures one training epoch.

l	CIFAR-10
1	6.69
2	5.43
3	5.65
4	5.93

Table 5.3: Test error (%, median over 5 runs) on CIFAR-10 of WRN-40-2 (2.2M) with various l.

Type of convolutions in a block

We start by reporting results using trained networks with different block types B (reported results are on CIFAR-10). We used WRN-40-2 for blocks $B(1,3,1)$, $B(3,1)$, $B(1,3)$ and $B(3,1,1)$ as these blocks have only one 3×3 convolution. To keep the number of parameters comparable we trained other networks with less layers: WRN-28-2-$B(3,3)$ and WRN-22-2-$B(3,1,3)$. We provide the results including test accuracy in median over 5 runs and time per training epoch in the table 5.2. Block $B(3,3)$ turned out to be the best by a little margin, and $B(3,1)$ with $B(3,1,3)$ are very close to $B(3,3)$ in accuracy having less parameters and less layers. $B(3,1,3)$ is faster than others by a small margin.

Based on the above, blocks with comparable number of parameters turned out to give more or less the same results. Due to this fact, we hereafter restrict our attention to only WRNs with 3×3 convolutions so as to be also consistent with other methods.

Number of convolutions per block

We next proceed with the experiments related to varying the deepening factor l (which represents the number of convolutional layers per block). We show indicative results in table 5.3, where in this case we took WRN-40-2 with 3×3 convolutions and trained several networks with different deepening factor $l \in [1,2,3,4]$, same number of parameters (2.2×10^6) and same number of convolutional layers.

As can be noticed, $B(3,3)$ turned out to be the best, whereas $B(3,3,3)$ and $B(3,3,3,3)$ had the worst performance. We speculate that this is probably due to the increased difficulty in optimization as a result of the decreased number of residual connections in the last two cases. Furthermore, $B(3)$ turned out to

depth	k	# params	CIFAR-10	CIFAR-100
40	1	0.6M	6.85	30.89
40	2	2.2M	5.33	26.04
40	4	8.9M	4.97	22.89
40	8	35.7M	4.66	-
28	10	36.5M	**4.17**	20.50
28	12	52.5M	4.33	**20.43**
22	8	17.2M	4.38	21.22
22	10	26.8M	4.44	20.75
16	8	11.0M	4.81	22.07
16	10	17.1M	4.56	21.59

Table 5.4: Test error (%) of various wide networks on CIFAR-10 and CIFAR-100 (ZCA preprocessing).

be quite worse. The conclusion is that $B(3,3)$ is optimal in terms of number of convolutions per block. For this reason, in the remaining experiments we only consider wide residual networks with a block of type $B(3,3)$.

Width of residual blocks

As we try to increase widening parameter k we have to decrease total number of layers. To find an optimal ratio we experimented with k from 2 to 12 and depth from 16 to 40. The results are presented in table 5.4. As can be seen, all networks with 40, 22 and 16 layers see consistent gains when width is increased by 1 to 12 times. On the other hand, when keeping the same fixed widening factor $k = 8$ or $k = 10$ and varying depth from 16 to 28 there is a consistent improvement, however when we further increase depth to 40 accuracy decreases (e.g. WRN-40-8 loses in accuracy to WRN-22-8).

We show additional results in table 5.5 where we compare thin and wide residual networks. As can be observed, wide WRN-40-4 can be compared to thin ResNet-1001 as they achieve approximately the same accuracy on CIFAR-10 and CIFAR-100. It is interesting that they have comparable number of parameters, 8.9×10^6 and 10.2×10^6, suggesting that depth does not add regularization effects compared to width at this level. As we show further in benchmarks, WRN-40-4 is 8 times faster to train, so evidently depth to width ratio in the original thin residual networks is far from optimal.

Also, wide WRN-28-10 outperforms thin ResNet-1001 by 0.8% (with the same minibatch size during training) on CIFAR-10 and 2.8% on CIFAR-100, having 36 times less layers (see table 5.5). We note that the result of 4.64% with ResNet-1001 was obtained with batch size 64, whereas we use a batch size

128 in all of our experiments (i.e., all other results reported in table 5.5 are with batch size 128). Training curves for these networks are presented in Figure 5.2.

Despite previous arguments that depth gives regularization effects and width causes network to overfit, we successfully train networks with 5 times more parameters than ResNet-1001. Wide WRN-28-12 (table 5.4) has 52.5×10^6 parameters and outperforms ResNet-1001 (table 5.5) by a significant margin.

	depth-k	# params	CIFAR-10	CIFAR-100
NIN Lin *et al.* (2013)			8.81	35.67
DSN Lee *et al.* (2014)			8.22	34.57
FitNet Romero *et al.* (2014)			8.39	35.04
Highway Srivastava *et al.* (2015)			7.72	32.39
ELU Clevert *et al.* (2015)			6.55	24.28
original-ResNetHe *et al.* (2016a)	110	1.7M	6.43	25.16
	1202	10.2M	7.93	27.82
stoc-depthHuang *et al.* (2016)	110	1.7M	5.23	24.58
	1202	10.2M	4.91	-
pre-act-ResNetHe *et al.* (2016b)	110	1.7M	6.37	-
	164	1.7M	5.46	24.33
	1001	10.2M	4.92(4.64)	22.71
WRN (ours)	40-4	8.7M	4.65	21.8
	16-8	11.0M	4.6	21.5
	28-10	36.5M	**4.15**	**19.92**

Table 5.5: Test error of different methods on CIFAR-10 and CIFAR-100 with moderate data augmentation (flip/translation) and mean/std normalzation. We don't use dropout for these results. In the second column k is a widening factor. Results for He *et al.* (2016b) are shown with minibatch size 128 (as ours), and 64 in parenthesis. Our results were obtained by computing median over 5 runs.

To be able to directly compare to original ResNet and follow-ups, we removed whitening preprocessing and trained WRN on mean/std normalized data, and updated tables 5.5 and 5.6. To our surprise this gave slightly better results. We further found out that mean/std preprocessing allows training wider and deeper networks with better accuracy, and achieved 18.5% on CIFAR-100 with WRN-40-10 with 80M parameters, giving total improvement of 4.2% over ResNet-1001.

To summarize:

- widening consistently improves performance across residual networks of different depth;
- increasing both depth and width helps until the number of parameters becomes too high and stronger regularization is needed;

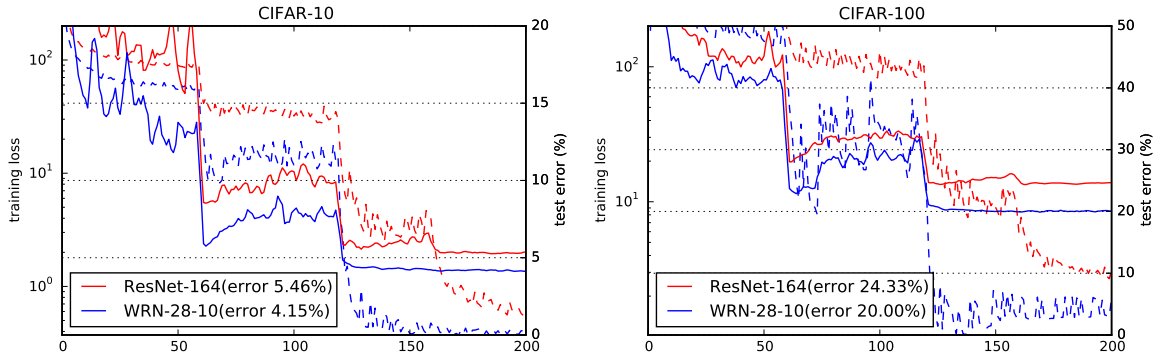


Figure 5.2: Training curves for thin and wide residual networks on CIFAR-10 and CIFAR-100. Solid lines denote test error (y-axis on the right), dashed lines denote training loss (y-axis on the left).

- there doesn't seem to be a regularization effect from very high depth in residual networks as wide networks with the same number of parameters as thin ones can learn same or better representations. Furthermore, wide networks can successfully learn with a 2 or more times larger number of parameters than thin ones, which would require doubling the depth of thin networks, making them infeasibly expensive to train.

Dropout in residual blocks

We trained networks with dropout inserted into residual block between convolutions on all datasets. We used cross-validation to determine dropout probability values, 0.3 on CIFAR and 0.4 on SVHN. Also, we didn't have to increase number of training epochs compared to baseline networks without dropout.

Dropout decreases test error on CIFAR-10 and CIFAR-100 by 0.16% and 0.48% correnspondingly (over median of 5 runs and mean/std preprocessing) with WRN-28-10, and gives improvements with other ResNets as well (table 5.6. To our knowledge, that's the first result to approach 20% error on CIFAR-100, even outperforming methods with heavy data augmentation. There is a noticeable drop in accuracy with WRN-16-4 on CIFAR which we speculate is due to the relatively small number of parameters.

We notice a disturbing effect in residual network training after the first learning rate drop when both loss and validation error suddenly start to go up and oscillate on high values until the next learning rate drop. We found out that it is caused by weight decay, however making it lower leads to a significant drop in accuracy. Interestingly, dropout partially removes this effect in most cases, see figures 5.2, 5.3.

The effect of dropout becomes more evident on SVHN. This is probably due to the fact that we don't do any data augmentation and batch normalization overfits, so dropout adds a regularization effect.

depth	k	dropout	CIFAR-10	CIFAR-100	SVHN
16	4		5.02	24.03	1.85
16	4	✓	5.24	23.91	1.64
28	10		4.15	19.92	-
28	10	✓	**3.99**	**19.44**	-
52	1		6.43	29.89	2.08
52	1	✓	6.28	29.78	1.70

Table 5.6: Effect of dropout in residual block. (mean/std preprocessing, CIFAR numbers are based on median of 5 runs)

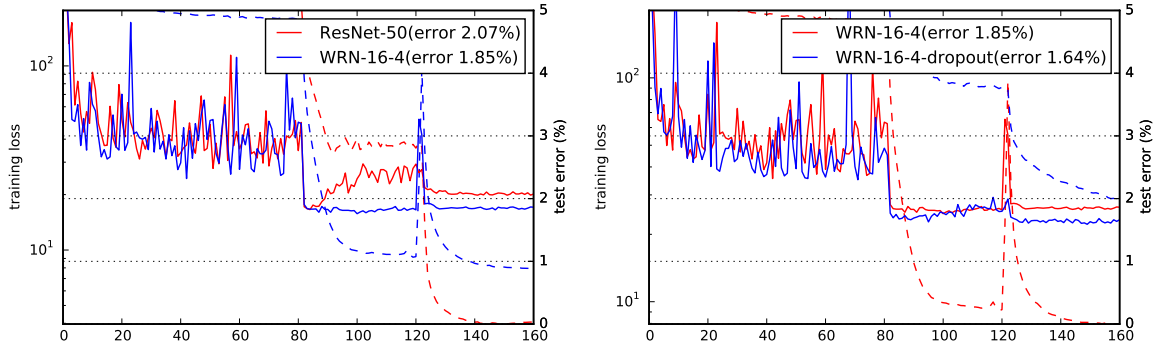


Figure 5.3: Training curves for SVHN. On the left: thin and wide networks, on the right: effect of dropout. Solid lines denote test error (y-axis on the right), dashed lines denote training loss (y-axis on the left).

Evidence for this can be found on training curves in figure 5.3 where the loss without dropout drops to very low values. The results are presented in table 5.6. We observe significant improvements from using dropout on both thin and wide networks. Thin 50-layer deep network even outperforms thin 152-layer deep network with stochastic depth Huang *et al.* (2016). We additionally trained WRN-16-8 with dropout on SVHN, which achieves impressive 1.54% on SVHN - the best published result to our knowledge. Without dropout it only achieves 1.81%.

Overall, despite the arguments of combining with batch normalization, dropout shows itself as an effective techique of regularization of thin and wide networks. It can be used to further improve results from widening, while also being complementary to it.

ImageNet

We first experiment with non-bottleneck ResNet-18 and ResNet-34, trying to gradually increase their width from 1.0 to 4.0. The results are shown in table 5.7. Increasing width gradually increases accuracy of both netowrks, and networks with the comparable number of parameters achieve similar results,

despite having different depth. Althouth these networks have a large number of parameters, they are outperfomed by bottleneck networks, which might be caused by two reasons: either bottleneck architecture is simply better suited for ImageNet classification task, or this more complex task needs a deeper network. To test this, we took the ResNet-50, and tried to make it wider by increasing inner 3×3 layer width. With widening factor of 2.0 WRN-50-2 outperforms ResNet-152 having 3 times less layers, and being significantly faster. WRN-50-2 is only slightly worse and almost $2\times$ faster than the best-performing pre-activation ResNet-200, althouth having slightly more parameters (table 5.8). In general, we find that, unlike CIFAR, ImageNet networks need more width at the same depth to achieve the same accuracy. It is however clear that it is unnecessary to have residual networks with more than 50 layers due to computational reasons.

We didn't try to train bigger bottleneck networks as 8-GPU machines are needed for that.

width		1.0	2.0	3.0	4.0
WRN-18	top1,top5	30.4, 10.93	27.06, 9.0	25.58, 8.06	24.06, 7.33
	#parameters	11.7M	25.9M	45.6M	101.8M
WRN-34	top1,top5	26.77, 8.67	24.5, 7.58	23.39, 7.00	
	#parameters	21.8M	48.6M	86.0M	

Table 5.7: ILSVRC-2012 validation error (single crop) of non-bottleneck ResNets with various width. Networks with the comparable number of parameters achieve similar accuracy, despite having 2 times less layers.

Model	top-1 err, %	top-5 err, %	#params	time/batch 16
ResNet-50	24.01	7.02	25.6M	49
ResNet-101	22.44	6.21	44.5M	82
ResNet-152	22.16	6.16	60.2M	115
WRN-50-2	21.9	6.03	68.9M	93
pre-ResNet-200	21.66	5.79	64.7M	154

Table 5.8: ILSVRC-2012 validation error (single crop) of bottleneck ResNets. Faster WRN-50-2 outperforms ResNet-152 having 3 times less layers, and stands close to pre-ResNet-200.

We also used WRN-34-2 to participate in COCO 2016 object detection challenge, using a combination of MultiPathNet Zagoruyko *et al.* (2016) and LocNet Gidaris and Komodakis (2016). Despite having only 34 layers, this system achieves state-of-the-art single model performance of 35.2 mAP, outperforming even ResNet-152 and Inception-v4-based networks.

Computational efficiency

Thin and deep residual networks with small kernels are against the nature of GPU computations because of their sequential structure. Increasing width helps effectively balance computations in much more optimal way, so that wide networks are many times more efficient than thin ones as our benchmarks show. We use cudnn v5 and Titan X to measure forward+backward update times with minibatch size 32 for several networks, the results are in the figure 5.4. We show that our best CIFAR wide WRN-28-10 is 1.6 times faster than thin ResNet-1001. Furthermore, wide WRN-40-4, which has approximately the same accuracy as ResNet-1001, is 8 times faster.

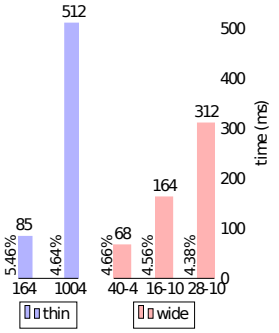


Figure 5.4: Time of forward+backward update per minibatch of size 32 for wide and thin networks. (x-axis denotes network depth and widening factor). Numbers beside bars indicate test error on CIFAR-10, on top - time (ms). Test time is a proportional fraction of these benchmarks. Note, for instance, that wide WRN-40-4 is 8 times faster than thin ResNet-1001 while having approximately the same accuracy.

5.3 Implicit skip-connections

As we show the first part of the chapter, the original motivation behind ResNet of training deeper networks does not actually hold true, and widening is more effective that deepening, meaning that there is no benefit from increasing depth to more than 50 layers. Additionally, widening networks are faster due to their parallel execution, whereas deeper networks need to be executed in a more sequential manner.

To summarize, deep networks with skip-connections have the following problems:

- Feature reuse problem: upper layers might not learn useful representations given previous activations;
- Widening is more effective than deepening: there is no benefit from increasing depth;
- Actual depth is not clear: it might be determined by the shortest path.

However, the features learned by such networks are generic, and they are able to train with massive number of parameters without negative effects of overfitting. We are thus interested in better understanding of networks with skip-connections, which would allow us to train very deep *plain* (without skip-connections) networks and benefits they could bring, such as higher parameter efficiency, better generalization, and improved computational efficiency.

Motivated by this, we propose a novel weight parameterization for neural networks, which we call Dirac parameterization, applicable to a wide range of network architectures. Furthermore, by use of the above parameterization, we propose novel plain VGG and ResNet-like architecture without explicit skip-connections, which we call DiracNet. These networks are able to train with hundreds of layers, surpass 1001-layer ResNet while having only 28-layers, and approach Wide ResNet (WRN) accuracy. We should note that we train DiracNets end-to-end, without any need of layer-wise pretraining. We believe that our work is an important step towards simpler and more efficient deep neural networks.

Overall, contributions of this part are the following:

- We propose generic Dirac weight parameterization, applicable to a wide range of neural network architectures;
- Our plain Dirac parameterized networks are able to train end-to-end with hundreds of layers. Furthermore, they are able to train with massive number of parameters and still generalize well without negative effects of overfitting;
- Dirac parameterization can be used in combination with explicit skip-connections like ResNet, in which case it eliminates the need of careful initialization.
- In a trained network Dirac-parameterized filters can be folded into a single vector, resulting in a simple and easily interpretable VGG-like network, a chain of convolution-ReLU pairs.

5.3.1 Dirac parameterization

Inspired from ResNet, we parameterize weights as a residual of Dirac function, instead of adding explicit skip connection. Because convolving any input with Dirac results in the same input, this helps propagate information deeper in the network. Similarly, on backpropagation it helps alleviate vanishing gradients problem.

Let $\boldsymbol{I}$ be the identity in algebra of discrete convolutional operators, i.e. convolving it with input $\boldsymbol{x}$ results in the same output $\boldsymbol{x}$ ($*$ denotes convolution):

$$\boldsymbol{I} * \boldsymbol{x} = \boldsymbol{x} \tag{5.1}$$

In two-dimensional case convolution might be expressed as matrix multiplication, so $\boldsymbol{I}$ is simply an identity matrix, or a Kronecker delta δ. We generalize this operator to the case of a convolutional layer, where input $\boldsymbol{x} \in \mathbb{R}^{M,N_1,N_2,...,N_L}$ (that consists of M channels of spatial dimensions (N_1, N_2, ..., N_L)) is convolved with weight $\hat{\boldsymbol{W}} \in \mathbb{R}^{M,M,K_1,K_2,...,K_L}$ (combining M filters[1]) to produce an output $\boldsymbol{y}$ of M channels, i.e. $\boldsymbol{y} = \hat{\boldsymbol{W}} * \boldsymbol{x}$. In this case we define Dirac delta $\boldsymbol{I} \in \mathbb{R}^{M,M,K_1,K_2,...,K_L}$, preserving eq. (5.1), as the following:

$$\boldsymbol{I}(i, j, l_1, l_2, \ldots, l_L) = \begin{cases} 1 & \text{if } i = j \text{ and } l_m \leq K_m \text{ for } m = 1..L, \\ 0 & \text{otherwise;} \end{cases} \tag{5.2}$$

Given the above definition, for a convolutional layer $\boldsymbol{y} = \hat{\boldsymbol{W}} * \boldsymbol{x}$ we propose the following parameterization for the weight $\hat{\boldsymbol{W}}$ (hereafter we omit bias for simplicity):

$$\boldsymbol{y} = \hat{\boldsymbol{W}} * \boldsymbol{x}, \tag{5.3}$$

$$\hat{\boldsymbol{W}} = \mathrm{diag}(\boldsymbol{a})\boldsymbol{I} + \boldsymbol{W}, \tag{5.4}$$

where $\boldsymbol{a} \in \mathbb{R}^M$ is scaling vector learned during training, and $\boldsymbol{W}$ is a weight vector. Each i-th element of $\boldsymbol{a}$ corresponds to scaling of i-th filter of $\boldsymbol{W}$. When all elements of $\boldsymbol{a}$ are close to zero, it reduces to a simple linear layer $\boldsymbol{W} * \boldsymbol{x}$. When they are higher than 1 and $\boldsymbol{W}$ is small, Dirac dominates, and the output is close to be the same as input.

We also use weight normalization Salimans and Kingma (2016) for $\boldsymbol{W}$, which we find useful for stabilizing training of very deep networks with more than 30 layers:

$$\hat{\boldsymbol{W}} = \mathrm{diag}(\boldsymbol{a})\boldsymbol{I} + \mathrm{diag}(\boldsymbol{b})\boldsymbol{W}_{\mathrm{norm}}, \tag{5.5}$$

where $\boldsymbol{b} \in \mathbb{R}^M$ is another scaling vector (to be learned during training), and $\boldsymbol{W}_{\mathrm{norm}}$ is a normalized weight vector where each filter is normalized by it's Euclidean norm. We initialize $\boldsymbol{a}$ to 1.0 and $\boldsymbol{b}$ to 0.1, and do not L^2-regularize them during training, as it would lead to degenerate solutions when their

[1] outputs are over the first dimension of $\hat{\boldsymbol{W}}$, inputs are over the second dimension of $\hat{\boldsymbol{W}}$

values are close to zero. We initialize $\boldsymbol{W}$ from normal distribution $\mathcal{N}(0, 1)$. Gradients of (5.5) can be easily calculated via chain-rule. We rely on automatic differentiation, available in all major modern deep learning frameworks (PyTorch, Tensorflow, Theano), to implement it.

Overall, this adds a negligible number of parameters to the network (just two scaling multipliers per channel) during training, which can be folded into filters at test time.

5.3.2 Connection between Dirac parameterization and residual block

Let us discuss the connection of Dirac parameterization to ResNet. Due to distributivity of convolution, eq. (5.3) can be rewritten to show that the skip-connection in Dirac parameterization is implicit (we omit $\boldsymbol{a}$ for simplicity):

$$\boldsymbol{y} = \sigma\big((\boldsymbol{I} + \boldsymbol{W}) * \boldsymbol{x}\big) = \sigma\big(\boldsymbol{x} + \boldsymbol{W} * \boldsymbol{x}\big), \tag{5.6}$$

where $\sigma(x)$ is a function combining nonlinearity and batch normalization. The skip connection in ResNet is explicit:

$$\boldsymbol{y} = \boldsymbol{x} + \sigma(\boldsymbol{W} * \boldsymbol{x})$$

This means that Dirac parameterization and ResNet differ only by the order of nonlinearities. Each delta parameterized layer adds complexity by having unavoidable nonlinearity, which is not the case for ResNet. Additionally, Dirac parameterization can be folded into a single weight vector on inference.

5.3.3 Experimental results

We adopt architecture similar to ResNet and VGG, and instead of explicit skip-connections use Dirac parameterization (see table 5.9). The architecture consists of three groups, where each group has $2N$ convolutional layers ($2N$ is used for easier comparison with basic-block ResNet and WRN, which have N blocks of pairs of convolutional layers per group). For simplicity we use max-pooling between groups to reduce spatial resolution. We also define width k as in WRN to control number of parameters.

We chose CIFAR and ImageNet for our experiments. As for baselines, we chose Wide ResNet with identity mapping in residual block He *et al.* (2016b) and basic block (two 3×3 convolutions per block). We used the same training hyperparameters as WRN for both CIFAR and ImageNet.

The experimental section is composed as follows. First, we provide a detailed experimental comparison between plain and plain-Dirac networks, and compare them with ResNet and WRN on CIFAR. Also,

we analyze evolution of scaling coefficients during training and their final values. Then, we present ImageNet results. Lastly, we apply Dirac parameterization to ResNet and show that it eliminates the need of careful initialization.

name	output size	layer type
`conv1`	32×32	[3×3, 16]
`group1`	32×32	$\left[3\times3,\ 16 \times 16k\right]\times2N$
`max-pool`	16×16	
`group2`	16×16	$\left[3\times3,\ 32k \times 32k\right]\times2N$
`max-pool`	8×8	
`group3`	8×8	$\left[3\times3,\ 64k \times 64k\right]\times2N$
`avg-pool`	1×1	$[8 \times 8]$

Table 5.9: Structure of DiracNets. Network width is determined by factor k. Groups of convolutions are shown in brackets as [kernel shape, number of input channels, number of output channels] where $2N$ is a number of layers in a group. Final classification layer and dimensionality changing layers are omitted for clarity.

Plain networks with Dirac parameterization

In this section we compare plain networks with plain DiracNets. To do that, we trained both with 10-52 layers and the same number of parameters at the same depth (fig. 5.5). As expected, at 10 and 16 layers there is no difficulty in training plain networks, and both plain and plain-Dirac networks achieve the same accuracy. After that, accuracy of plain networks quickly drops, and with 52 layers only achieves 88%, whereas for Dirac parameterized networks it keeps growing. DiracNet with 34 layers achieves 92.8% validation accuracy, whereas simple plain only 91.2%. Plain 100-layer network does not converge and only achieves 40% train/validation accuracy, whereas DiracNet achieves 92.4% validation accuracy.

Plain Dirac networks and residual networks

To compare plain Dirac parameterized networks with WRN we trained them with different width k from 1 to 4 and depth from 10 to 100 (fig. 5.5). As observed by WRN authors, accuracy of ResNet is mainly determined by the number of parameters, and we even notice that wider networks achieve better performance than deeper. DiracNets, however, benefit from depth, and deeper networks with the same accuracy as wider have less parameters. In general, DiracNets need more parameters than ResNet to achieve top accuracy, and we were able to achieve 95.25% accuracy with DiracNet-28-10 with 36.5M parameters, which is close to WRN-28-10 with 96.0% and 36.5M parameters as well. We do not observe validation accuracy degradation when increasing width, the networks still perform well

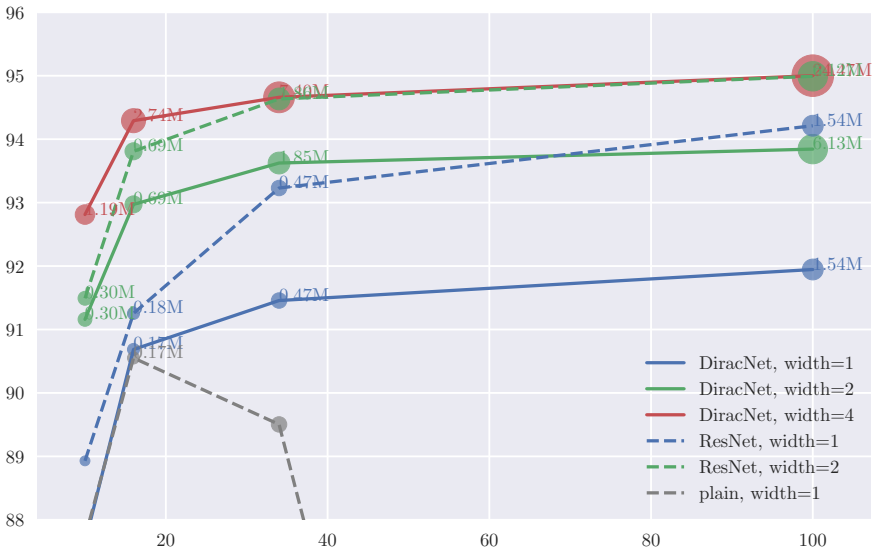


Figure 5.5: DiracNet and ResNet with different depth/width. Each circle area is proportional to number of parameters. DiracNet needs more width (i.e. parameters) to match ResNet accuracy. Accuracy is calculated as median of 5 runs.

	depth-width	# params	CIFAR-10	CIFAR-100
NIN Lin *et al.* (2013),			8.81	35.67
ELU Clevert *et al.* (2015),			6.55	24.28
VGG	16	20M	6.09±0.11	25.92±0.09
DiracNet (ours)	28-5	9.1M	5.16±0.14	23.44±0.14
	28-10	36.5M	4.75±0.16	21.54±0.18
ResNet	1001-1	10.2M	4.92	22.71
Wide ResNet	28-10	36.5M	**4.00**	**19.25**

Table 5.10: CIFAR performance of plain (top part) and residual (bottom part) networks on with horizontal flips and crops data augmentation. DiracNets outperform all other plain networks by a large margin, and approach residual architectures. No dropout it used. For VGG and DiracNets we report mean±std of 5 runs.

despite the massive number of parameters, just like WRN. Interestingly, plain DiracNet with only 28 layers is able to closely match ResNet with 1001 layers (table 5.10)

Analysis of scaling coefficients

As we leave $\boldsymbol{a}$ and $\boldsymbol{b}$ free of L^2-regularization, we can visualize significance of various layers and how it changes during training by plotting their averages $\bar{\boldsymbol{a}}$ and $\bar{\boldsymbol{b}}$, which we did for DiracNet-34 trained on CIFAR-10 on fig. 5.6. Interestingly, the behaviour changes from lower to higher groups of the network with increasing dimensionality. We also note that no layers exhibit degraded $\boldsymbol{a}$ to $\boldsymbol{b}$ ratio, meaning that

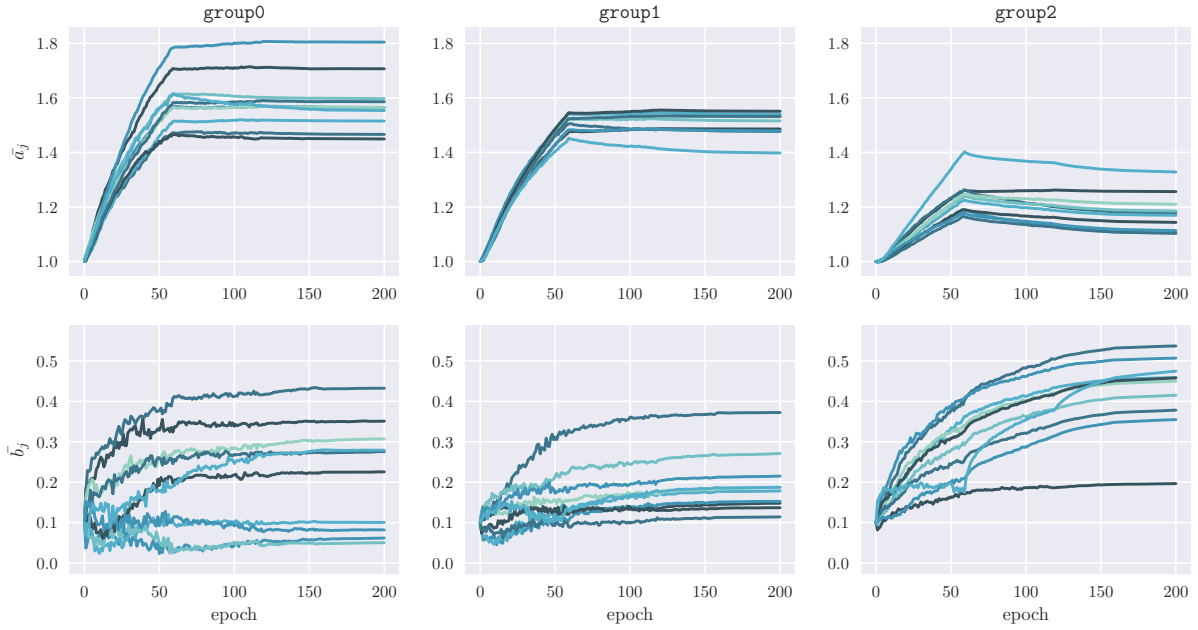


Figure 5.6: Average values of $\boldsymbol{a}$ and $\boldsymbol{b}$ during training for different layers of DiracNet-34. Deeper color means deeper layer in a group of blocks.

all layers are involved in training. We also investigate these ratios in individual feature planes, and find that the number of degraded planes is low too.

Dirac parameterization for ResNet weight initialization

As expected, Dirac parameterization does not bring accuracy improvements to ResNet on CIFAR, but eliminates the need of careful initialization. To test that, instead of usually used MSRA init He *et al.* (2015), we parameterize weights as:

$$\hat{\boldsymbol{W}} = \boldsymbol{I} + \boldsymbol{W},$$

omitting other terms of eq. (5.5) for simplicity, and initialize all weights from a normal distribution $\mathcal{N}(0, \sigma^2)$, ignoring filter shapes. Then, we vary σ and observe that ResNet-28 converges to the same validation accuracy with statistically insignificant deviations, even for very small values of σ such as 10^{-8}, and only gives slightly worse results when σ is around 1. It does not converge when all weights are zeros, as expected. Additionally, we tried to use the same orthogonal initialization as for DiracNet and vary it's scaling, in which case the range of the scaling gain is even wider.

	Network	# parameters	top-1 error	top-5 error
plain	VGG-CNN-S Chatfield *et al.* (2014)	102.9M	36.94	15.40
	VGG-16 Simonyan and Zisserman (2015)	138.4M	29.38	-
	DiracNet-18	11.7M	30.37	10.88
	DiracNet-34	21.8M	27.79	9.34
residual	ResNet-18 [our baseline]	11.7M	29.62	10.62
	ResNet-34 [our baseline]	21.8M	27.17	8.91

Table 5.11: Single crop top-1 and top-5 error on ILSVRC2012 validation set for plain (top) and residual (bottom) networks.

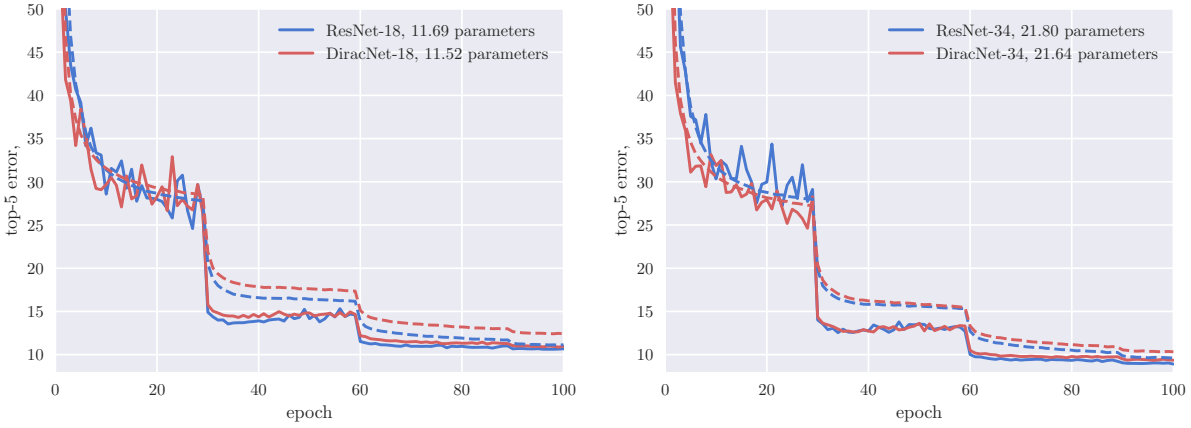


Figure 5.7: Convergence of DiracNet and ResNet on ImageNet. Training top-5 error is shown with dashed lines, validation - with solid. All networks are trained using the same optimization hyperparameters. DiracNet closely matches ResNet accuracy with the same number of parameters.

ImageNet results

We trained DiracNets with 18 and 34 layers and their ResNet equivalents on ILSVRC2012 image classification dataset. We used the same setup as for ResNet training, and kept the same number of blocks per groups. Unlike on CIFAR, DiracNet almost matches ResNet in accuracy (table 5.11), with very similar convergence curves (fig. 5.7) and the same number of parameters. As for simple plain VGG networks, DiracNets achieve same accuracy with 10 times less parameters, similar to ResNet.

5.4 Conclusions

In the first part of the chapter we presented a study on width of residual networks and showed state-of-the-art results on CIFAR-10, CIFAR-100, SVHN and significant improvements on ImageNet only due to increased width of residual networks. We show that wide networks with only 16 layers can

significantly outperform 1000-layer deep networks on CIFAR, as well as 50-layer outperform 152-layer on ImageNet, showing that the main power of residual networks is in residual blocks, and not in extreme depth as claimed earlier. Also, wide residual networks are several times faster to train. We think that these intriguing findings will help further advances in research in deep neural networks.

Motivated by the wide residual networks, in the second part we proposed Dirac-parameterized networks, a simple and efficient way to train very deep networks with nearly state-of-the-art accuracy. Even though they are able to successfully train with hundreds of layers, after a certain number of layers there seems to be very small or no benefit in terms of accuracy for both ResNets and DiracNets. This is likely caused by underuse of parameters in deeper layers, and both architectures are prone to this issue to a different extent.

Even though on large ImageNet dataset DiracNets are able to closely match ResNet in accuracy with the same number of parameters and a simpler architecture, they are significantly behind on smaller CIFAR datasets, which we think is due to lack of regularization, more important on small amounts of data. Due to use of weight normalization and free scaling parameters DiracNet is less regularized than ResNet, which we plan to investigate in future.

We also observe that DiracNets share the same property as WRN to train with massive number of parameters and still generalize well without negative effects of overfitting, which was initially thought was due to residual connections. We now hypothesize that it is due to a combination of SGD with momentum at high learning rate, which has a lot of noise, and stabilizing factors, such as residual or Dirac parameterization, batch normalization, etc.

Chapter 6

Improving convolutional neural networks via attention transfer

In this chapter we explore attention, which plays a critical role in human visual experience. Furthermore, it has recently been demonstrated that attention can also play an important role in the context of applying artificial neural networks to a variety of tasks from fields such as computer vision and NLP. In this work we show that, by properly defining attention for convolutional neural networks, we can actually use this type of information in order to significantly improve the performance of a student CNN network by forcing it to mimic the attention maps of a powerful teacher network. To that end, we propose several novel methods of transferring attention, showing consistent improvement across a variety of datasets and convolutional neural network architectures.

This chapter is based on Paying More Attention to Attention: Improving the Performance of Convolutional Neural Networks via Attention Transfer Zagoruyko and Komodakis (2017b).

6.1 Introduction

As humans, we need to pay attention in order to be able to adequately perceive our surroundings. Attention is therefore a key aspect of our visual experience, and closely relates to perception - we need to keep attention to build a visual representation, possessing detail and coherence.

As artificial neural networks became more popular in fields such as computer vision and natural language processing in the recent years, artificial attention mechanisms started to be developed as well. Artificial attention lets a system "attend" to an object to examine it with greater detail. It has also become a research tool for understanding mechanisms behind neural networks, similar to attention used in psychology.

One of the popular hypothesis there is that there are non-attentional and attentional perception processes. Non-attentional processes help to observe a scene in general and gather high-level information, which, when associated with other thinking processes, helps us to control the attention processes and navigate to a certain part of the scene. This implies that different observers — with different knowledge, different goals, and therefore different attentional strategies — can literally see the same scene differently. This brings us to the main topic of this chapter: how attention differs within artificial vision systems, and can we use attention information in order to improve the performance of convolutional neural networks ? More specifically, can a teacher network improve the performance of another student network by providing to it information about where it looks, i.e., about where it concentrates its attention into ?

To study these questions, one first needs to properly specify how attention is defined w.r.t. a given convolutional neural network. To that end, here we consider attention as a set of *spatial* maps that essentially try to encode on which spatial areas of the input the network focuses most for taking its output decision (e.g., for classifying an image), where, furthermore, these maps can be defined w.r.t. various layers of the network so that they are able to capture both low-, mid-, and high-level representation information. More specifically, in this work we define two types of spatial attention maps: *activation-based* and *gradient-based*. We explore how both of these attention maps change over various datasets and architectures, and show that these actually contain valuable information that can be used for significantly improving the performance of convolutional neural network architectures (of various types and trained for various different tasks). To that end, we propose several novel ways of transferring attention from a powerful teacher network to a smaller student network with the goal of improving the performance of the latter (Fig. 6.1).

(a) (b)

Figure 6.1: **(a)** An input image and a corresponding spatial attention map of a convolutional network that shows where the network focuses in order to classify the given image. Surely, this type of map must contain valuable information about the network. The question that we pose in this chapter is the following: can we use knowledge of this type to improve the training of CNN models ? **(b)** Schematic representation of attention transfer: a student CNN is trained so as, not only to make good predictions, but to also have similar spatial attention maps to those of an already trained teacher CNN.

To summarize, the contributions of this work are as follows:

- We propose attention as a mechanism of transferring knowledge from one network to another
- We propose the use of both activation-based and gradient-based spatial attention maps
- We show experimentally that our approach provides significant improvements across a variety of datasets and deep network architectures, including both residual and non-residual networks
- We show that activation-based attention transfer gives better improvements than full-activation transfer, and can be combined with knowledge distillation

The rest of the chapter is structured as follows: we first describe related work in section 6.2, we explain our approach for activation-based and gradient-based attention transfer in section 6.3, and then present experimental results for both methods in section 6.4. We conclude the chapter in section 6.5.

6.2 Related work

Early work on attention based tracking Larochelle and Hinton (2010), Denil *et al.* (2012) was motivated by human attention mechanism theories Rensink (2000) and was done via Restricted Bolzmann Machines. It was recently adapted for neural machine translation with recurrent neural networks, e.g. Bahdanau *et al.* (2014) as well as in several other NLP-related tasks. It was also exploited in computer-vision-related tasks such as image captioning Xu *et al.* (2015), visual question answering Yang *et al.*

(2016), as well as in weakly-supervised object localization Oquab *et al.* (2015) and classification Mnih *et al.* (2014), to mention a few characteristic examples. In all these tasks attention proved to be useful.

Visualizing attention maps in deep convolutional neural networks is an open problem. The simplest gradient-based way of doing that is by computing a Jacobian of network output w.r.t. input (this leads to attention visualization that are not necessarily class-discriminative), as for example in Simonyan *et al.* (2014). Another approach was proposed by Zeiler and Fergus (2014) that consists of attaching a network called "deconvnet" that shares weights with the original network and is used to project certain features onto the image plane. A number of methods was proposed to improve gradient-based attention as well, for example guided backpropagation Springenberg *et al.* (2015), adding a change in *ReLU* layers during calculation of gradient w.r.t. previous layer output. Attention maps obtained with guided backpropagation are non-class-discriminative too. Among existing methods for visualizing attention, we should also mention class activation maps Zhou *et al.* (2016), which are based on removing top average-pooling layer and converting the linear classification layer into a convolutional layer, producing attention maps per each class. A method combining both guided backpropagation and CAM is Grad-CAM by Selvaraju *et al.* (2017), adding image-level details to class-discriminative attention maps.

Knowledge distillation with neural networks was pioneered by Hinton *et al.* (2015); Bucila *et al.* (2006), which is a transfer learning method that aims to improve the training of a student network by relying on knowledge borrowed from a powerful teacher network. Although in certain special cases shallow networks had been shown to be able to approximate deeper ones without loss in accuracy Lei and Caruana (2014), later work related to knowledge distillation was mostly based on the assumption that deeper networks always learn better representations. For example, FitNets Romero *et al.* (2014) tried to learn a thin deep network using a shallow one with more parameters. The introduction of highway Srivastava *et al.* (2015) and later residual networks He *et al.* (2016a) allowed training very deep architectures with higher accuracy, and generality of these networks was experimentally showed over a large variety of datasets. Although the main motivation for residual networks was increasing depth, it was later shown by Zagoruyko and Komodakis (2016b) that, after a certain depth, the improvements came mostly from increased capacity of the networks, i.e. number of parameters (for instance, a wider deep residual network with only 16 layers was shown that it could learn as good or better representations as very thin 1000 layer one, provided that they were using comparable number of parameters).

Due to the above fact and due to that thin deep networks are less parallelizable than wider ones, we think that knowledge transfer needs to be revisited, and take an opposite to FitNets approach - we try to learn less deep student networks. Our attention maps used for transfer are similar to both gradient-based and

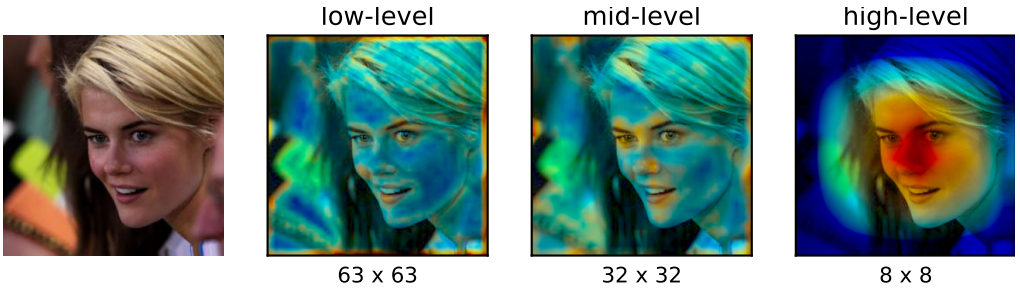


Figure 6.2: Sum of absolute values attention maps F_{sum} over different levels of a network trained for face recognition. Mid-level attention maps have higher activation level around eyes, nose and lips, high-level activations correspond to the whole face.

activation-based maps mentioned above, which play a role similar to "hints" in FitNets, although we don't introduce new weights.

6.3 Attention transfer

In this section we explain the two methods that we use for defining the spatial attention maps of a convolutional neural network as well as how we transfer attention information from a teacher to a student network in each case.

6.3.1 Activation-based attention transfer

Let us consider a CNN layer and its corresponding activation tensor $A \in R^{C\times H\times W}$, which consists of C feature planes with spatial dimensions $H \times W$. An activation-based mapping function $\mathcal{F}$ (w.r.t. that layer) takes as input the above 3D tensor A and outputs a spatial attention map, i.e., a flattened 2D tensor defined over the spatial dimensions, or

$$\mathcal{F} : R^{C\times H\times W} \rightarrow R^{H\times W} \quad .$$

To define such a spatial attention mapping function, the implicit assumption that we make in this section is that the absolute value of a hidden neuron activation (that results when the network is evaluated on given input) can be used as an indication about the importance of that neuron w.r.t. the specific input. By considering, therefore, the absolute values of the elements of tensor A, we can construct a spatial attention map by computing statistics of these values across the channel dimension (see Fig. 6.3). More specifically, in this work we will consider the following activation-based spatial attention maps:

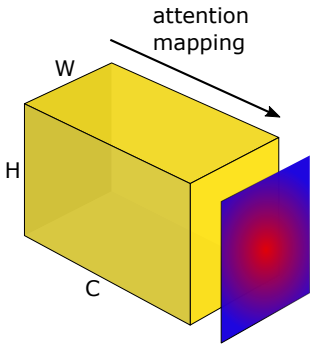


Figure 6.3: Attention mapping over feature dimension

- sum of absolute values: $F_{\text{sum}}(A) = \sum_{i=1}^{C} |A_i|$
- sum of absolute values raised to the power of p (where $p > 1$): $F_{\text{sum}}^{p}(A) = \sum_{i=1}^{C} |A_i|^p$
- max of absolute values raised to the power of p (where $p > 1$): $F_{\text{max}}^{p}(A) = \max_{i=1,C} |A_i|^p$

where $A_i = A(i, :, :)$ (using Matlab notation), and max, power and absolute value operations are elementwise (e.g. $|A_i|^p$ is equivalent to `abs(A_i).^p` in Matlab notation).

We visualized activations of various networks on several datasets, including ImageNet classification and localization, COCO object detection, face recognition, and fine-grained recognition. We were mostly focused on modern architectures without top dense linear layers, such as Network-In-Network, ResNet and Inception, which have streamlined convolutional structure. We also examined networks of the same architecture, width and depth, but trained with different frameworks with significant difference in performance. We found that the above statistics of hidden activations not only have spatial correlation with predicted objects on image level, but these correlations also tend to be higher in networks with higher accuracy, and stronger networks have peaks in attention where weak networks don't (e.g., see Fig. 6.4). Furthermore, attention maps focus on different parts for different layers in the network. In the first layers neurons activation level is high for low-level gradient points, in the middle it is higher for the most discriminative regions such as eyes or wheels, and in the top layers it reflects full objects. For example, mid-level attention maps of a network trained for face recognition Parkhi *et al.* (2015) will have higher activations around eyes, nose and lips, and top level activation will correspond to full face (Fig. 6.2).

Concerning the different attention mapping functions defined above, these can have slightly different properties. E.g.:

- Compared to $F_{\text{sum}}(A)$, the spatial map $F_{\text{sum}}^{p}(A)$ (where $p > 1$) puts more weight to spatial locations that correspond to the neurons with the highest activations, i.e., puts more weight to the

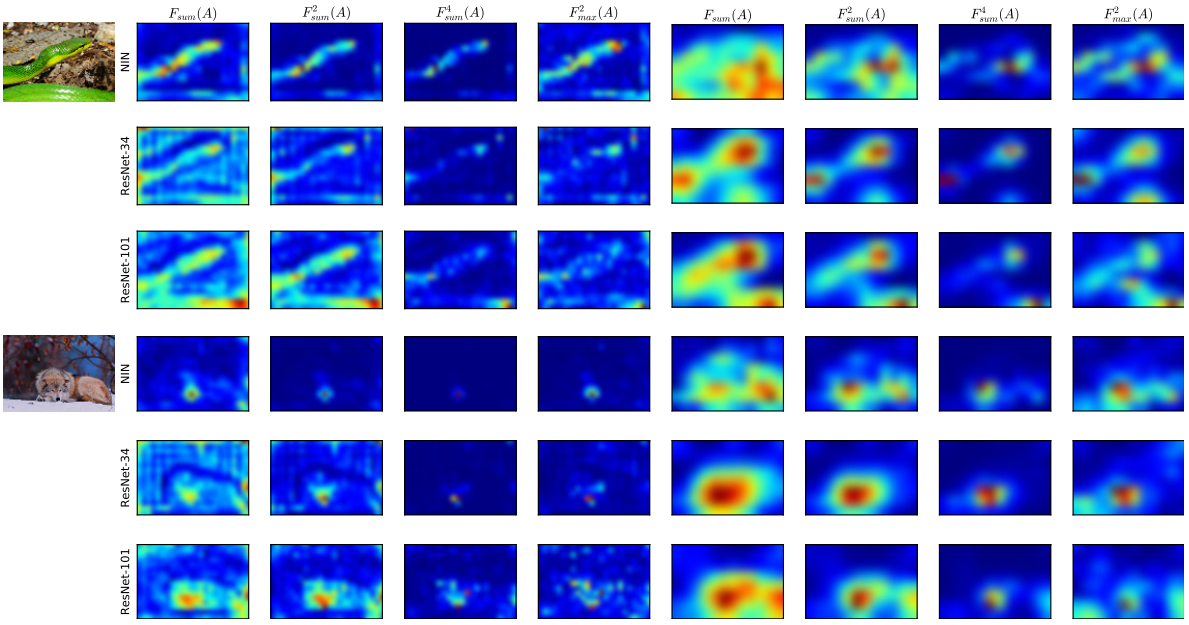


Figure 6.4: Activation attention maps for various ImageNet networks. Network-In-Network (62% top-1 val accuracy), ResNet-34 (73% top-1 val accuracy), ResNet-101 (77.3% top-1 val accuracy). Left part: mid-level activations, right part: top-level pre-softmax activations

most discriminative parts (the larger the p the more focus is placed on those parts with highest activations).

- Furthermore, among all neuron activations corresponding to the same spatial location, $F^p_{\max}(A)$ will consider only one of them to assign a weight to that spatial location (as opposed to $F^p_{\text{sum}}(A)$ that will favor spatial locations that carry multiple neurons with high activations).

To further illustrate the differences of these functions we visualized attention maps of 3 networks with sufficient difference in classification performance: Network-In-Network (62% top-1 val accuracy), ResNet-34 (73% top-1 val accuracy) and ResNet-101 (77.3% top-1 val accuracy). In each network we took last pre-downsampling activation maps, on the left for mid-level and on the right for top pre-average pooling activations in fig. 6.4. Top-level maps are blurry because their original spatial resolution is 7×7. It is clear that most discriminative regions have higher activation levels, e.g. face of the wolf, and that shape details disappear as the parameter p (used as exponent) increases.

In attention transfer, given the spatial attention maps of a teacher network (computed using any of the above attention mapping functions), the goal is to train a student network that will not only make correct predictions but will also have attentions maps that are similar to those of the teacher. In general, one can place transfer losses w.r.t. attention maps computed across several layers. For instance, in the case of

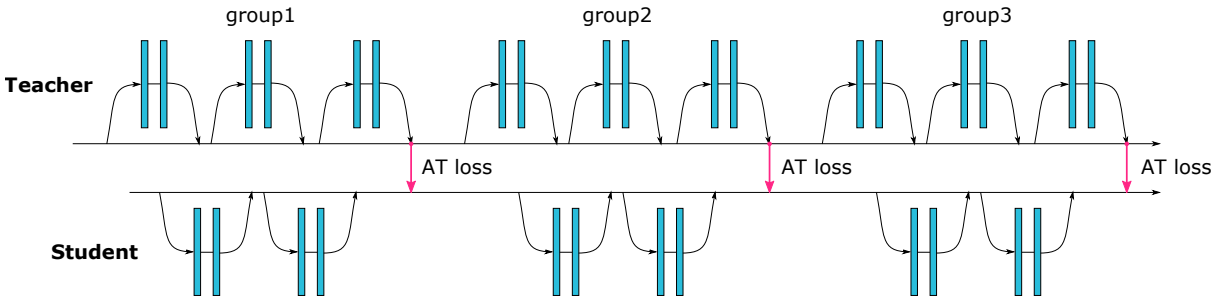


Figure 6.5: Schematics of teacher-student attention transfer for the case when both networks are residual, and the teacher is deeper.

ResNet architectures, one can consider the following two cases, depending on the depth of teacher and student:

- Same depth: possible to have attention transfer layer after every residual block
- Different depth: have attention transfer on output activations of each group of residual blocks

Similar cases apply also to other architectures (such as NIN, in which case a group refers to a block of a 3×3, 1×1, 1×1 convolutions). In fig. 6.5 we provide a schematic illustration of the different depth case for residual network architectures.

Without loss of generality, we assume that transfer losses are placed between student and teacher attention maps of same spatial resolution, but, if needed, attention maps can be interpolated to match their shapes. Let S, T and $\mathbf{W}_S$, $\mathbf{W}_T$ denote student, teacher and their weights correspondingly, and let $\mathcal{L}(\mathbf{W}, x)$ denote a standard cross entropy loss. Let also $\mathcal{I}$ denote the indices of all teacher-student activation layer pairs for which we want to transfer attention maps. Then we can define the following total loss:

$$\mathcal{L}_{AT} = \mathcal{L}(\mathbf{W}_S, x) + \frac{\beta}{2} \sum_{j \in \mathcal{I}} \| \frac{Q_S^j}{\|Q_S^j\|_2} - \frac{Q_T^j}{\|Q_T^j\|_2} \|_p \ , \tag{6.1}$$

where $Q_S^j = vec(F(A_S^j))$ and $Q_T^j = vec(F(A_T^j))$ are respectively the j-th pair of student and teacher attention maps in vectorized form, and p refers to norm type (in the experiments we use $p = 2$). As can be seen, during attention transfer we make use of l_2-normalized attention maps, i.e., we replace each vectorized attention map Q with $\frac{Q}{\|Q\|_2}$ (l_1 normalization could be used as well). It is worth emphasizing that normalization of attention maps is important for the success of the student training.

Attention transfer can also be combined with knowledge distillation Hinton *et al.* (2015), in which case an additional term (corresponding to the cross entropy between softened distributions over labels of teacher and student) simply needs to be included to the above loss. When combined, attention transfer

adds very little computational cost, as attention maps for teacher can be easily computed during forward propagation, needed for distillation.

6.3.2 Gradient-based attention transfer

In this case we define attention as gradient w.r.t. input, which can be viewed as an input sensitivity map Simonyan *et al.* (2014), i.e., attention at an input spatial location encodes how sensitive the output prediction is w.r.t. changes at that input location (e.g., if small changes at a pixel can have a large effect on the network output then it is logical to assume that the network is "paying attention" to that pixel). Let's define the gradient of the loss w.r.t input for teacher and student as:

$$J_S = \frac{\partial}{\partial x}\mathcal{L}(\mathbf{W_S}, x), J_T = \frac{\partial}{\partial x}\mathcal{L}(\mathbf{W_T}, x)$$

Then if we want student gradient attention to be similar to teacher attention, we can minimize a distance between them (here we use l_2 distance but other distances can be employed as well):

$$\mathcal{L}_{AT}(\mathbf{W_S}, \mathbf{W_T}, x) = \mathcal{L}(\mathbf{W_S}, x) + \frac{\beta}{2}||J_S - J_T||_2$$

As $\mathbf{W}_T$ and x are given, to get the needed derivative w.r.t. $\mathbf{W}_S$:

$$\frac{\partial}{\partial \mathbf{W_S}}\mathcal{L}_{AT} = \frac{\partial}{\partial \mathbf{W_S}}\mathcal{L}(\mathbf{W_S}, x) + \beta(J_S - J_T)\frac{\partial^2}{\partial \mathbf{W_S}\partial x}\mathcal{L}(\mathbf{W_S}, x) \tag{6.2}$$

So to do an update we first need to do forward and back propagation to get J_S and J_T, compute the second error $\frac{\beta}{2}||J_S - J_T||_2$ and propagate it second time. The second propagation is similar to forward propagation in this case, and involves second order mixed partial derivative calculation $\frac{\partial^2}{\partial W_S \partial x}$. The above computation is similar to the double backpropagation technique developed by Drucker and LeCun (1992) (where the l_2 norm of the gradient w.r.t. input is used as regularizer). Furthermore, it can be implemented efficiently in a framework with automatic differentiation support, even for modern architectures with sophisticated graphs. The second backpropagation has approximately the same cost with first backpropagation, excluding forward propagation.

We also propose to enforce horizontal flip invariance on gradient attention maps. To do that we propagate horizontally flipped images as well as originals, backpropagate and flip gradient attention maps back.

We then add l_2 losses on the obtained attentions and outputs, and do second backpropagation:

$$\mathcal{L}_{sym}(\mathbf{W}, x) = \mathcal{L}(\mathbf{W}, x) + \frac{\beta}{2}||\frac{\partial}{\partial x}\mathcal{L}(\mathbf{W}, x) - \text{flip}(\frac{\partial}{\partial x}\mathcal{L}(\mathbf{W}, \text{flip}(x)))||_2 \ , \tag{6.3}$$

where $\text{flip}(x)$ denotes the flip operator. This is similar to Group Equivariant CNN approach by Cohen and Welling (2016), however it is not a hard constraint. We experimentally find that this has a regularization effect on training.

We should note that in this work we consider only gradients w.r.t. the input layer, but in general one might have the proposed attention transfer and symmetry constraints w.r.t. higher layers of the network.

6.4 Experimental results

In the following section we explore attention transfer on various image classification datasets. We split the section in two parts, in the first we include activation-based attention transfer and gradient-based attention transfer experiments on CIFAR, and in the second activation-based attention transfer experiments on larger datasets. For activation-based attention transfer we used Network-In-Network Lin *et al.* (2013) and ResNet-based architectures (including the recently introduced Wide Residual Networks (WRN) Zagoruyko and Komodakis (2016b)), as they are most performant and set strong baselines in terms of number of parameters compared to AlexNet or VGG, and have been explored in various papers across small and large datasets. On Scenes, CUB and ImageNet we experimented with ResNet-18 and ResNet-34. As for gradient-based attention, we constrained ourselves to Network-In-Network without batch normalization and CIFAR dataset, due to the need of complex automatic differentiation.

6.4.1 CIFAR experiments

We start with CIFAR dataset which has small 32×32 images, and after downsampling top activations have even smaller resolution, so there is not much space for attention transfer. Interestingly, even under this adversarial setting, we find that attention transfer seems to give reasonable benefits, offering in all cases consistent improvements. We use horizontal flips and random crops data augmentations, and all networks have batch normalization. We find that ZCA whitening has negative effect on validation accuracy, and omit it in favor of simpler meanstd normalization. We raise Knowledge Distillation (KD) temperature for ResNet transfers to 4, and use $\alpha = 0.9$ (see Hinton *et al.* (2015) for an explanation of these parameters).

6.4.1.1 Activation-based attention transfer

Results of attention transfer (using F^2_{sum} attention maps) for various networks on CIFAR-10 can be found in table 6.1. We experimented with teacher/student having the same depth (WRN-16-2/WRN-16-1), as well as different depth (WRN-40-1/WRN-16-1, WRN-40-2/WRN-16-2). In all combinations, attention transfer (AT) shows significant improvements, which are also higher when it is combined with knowledge distillation (AT+KD).

student	teacher	student	AT	F-ActT	KD	AT+KD	teacher
NIN-thin, 0.2M	NIN-wide, 1M	9.38	8.93	9.05	8.55	8.33	7.28
WRN-16-1, 0.2M	WRN-16-2, 0.7M	8.77	7.93	8.51	7.41	7.51	6.31
WRN-16-1, 0.2M	WRN-40-1, 0.6M	8.77	8.25	8.62	8.39	8.01	6.58
WRN-16-2, 0.7M	WRN-40-2, 2.2M	6.31	5.85	6.24	6.08	5.71	5.23

Table 6.1: Activation-based attention transfer (AT) with various architectures on CIFAR-10. Error is computed as median of 5 runs with different seed. F-ActT means full-activation transfer (see §6.4.1.2).

To verify if having at least one activation-based attention transfer loss per group in WRN transfer is important, we trained three networks with only one transfer loss per network in `group1`, `group2` and `group3` separately, and compared to a network trained with all three losses. The corresponding results were 8.11, 7.96, 7.97 (for the separate losses) and 7.93 for the combined loss (using WRN-16-2/WRN-16-1 as teacher/student pair). Each loss provides some additional degree of attention transfer.

We also explore which attention mapping functions tend to work best using WRN-16-1 and WRN-16-2 as student and teacher networks respectively (table 6.2). Interestingly, sum-based functions work very similar, and better than max-based ones. From now on, we will use sum of squared attention mapping function F^2_{sum} for simplicity. As for parameter β in eq. 6.1, it usually varies about 0.1, as we set it to 10^3 divided by number of elements in attention map and batch size for each layer. In case of combinining AT with KD we decay it during traning in order to simplify learning harder examples.

6.4.1.2 Activation-based AT vs. transferring full activation

To check if transferring information from full activation tensors is more beneficial than from attention maps, we experimented with FitNets-style hints using l_2 losses on full activations directly, with 1×1 convolutional layers to match tensor shapes, and found that improvements over baseline student were minimal (see column F-ActT in table 6.1). For networks of the same width different depth we tried to regress directly to activations, without 1×1 convolutions. We also use l_2 normalization before

transfer losses, and decay β in eq. 6.1 during training as these give better performance. We find that AT, as well as full-activation transfer, greatly speeds up convergence, but AT gives much better final accuracy improvement than full-activation transfer (see fig. 6.7b, Appendix). It seems quite interesting that attention maps carry information that is more important for transfer than full activations.

attention mapping function	error
no attention transfer	8.77
F_{sum}	7.99
F^2_{sum}	7.93
F^4_{sum}	8.09
F^1_{max}	8.08

Table 6.2: Test error of WRN-16-2/WRN-16-1 teacher/student pair for various attention mapping functions. Median of 5 runs test errors are reported.

norm type	error
baseline (no attention transfer)	13.5
min-l_2 Drucker and LeCun (1992)	12.5
grad-based AT	12.1
KD	12.1
symmetry norm	11.8
activation-based AT	**11.2**

Table 6.3: Performance of various gradient-based attention methods on CIFAR-10. Baseline is a thin NIN network with 0.2M parameters (trained only on horizontally flipped augmented data and without batch normalization), min-l_2 refers to using l_2 norm of gradient w.r.t. input as regularizer, symmetry norm - to using flip invariance on gradient attention maps (see eq. 6.3), AT - to attention transfer, and KD - to Knowledge Distillation (both AT and KD use a wide NIN of 1M parameters as teacher).

6.4.1.3 Gradient-based attention transfer

For simplicity we use thin Network-In-Network model in these experiments, and don't apply random crop data augmentation with batch normalization, just horizontal flips augmentation. We also only use deterministic algorithms and sampling with fixed seed, so reported numbers are for single run experiments. We find that in this setting network struggles to fit into training data already, and turn off weight decay even for baseline experiments. In future we plan to explore gradient-based attention for teacher-student pairs that make use of batch normalization, because it is so far unclear how batch normalization should behave in the second backpropagation step required during gradient-based attention transfer (e.g., should it contribute to batch normalization parameters, or is a separate forward propagation with fixed parameters needed).

We explored the following methods:

- Minimizing l_2 norm of gradient w.r.t. input, i.e. the double backpropagation method Drucker and LeCun (1992);

- Symmetry norm on gradient attention maps (see eq. 6.3);
- Student-teacher gradient-based attention transfer;
- Student-teacher activation-based attention transfer.

Results for various methods are shown in table 6.3. Interestingly, just minimizing l_2 norm of gradient already works pretty well. Also, symmetry norm is one the best performing attention norms, which we plan to investigate in future on other datasets as well. We also observe that, similar to activation-based attention transfer, using gradient-based attention transfer leads to improved performance. We also trained a network with activation-based AT in the same training conditions, which resulted in the best performance among all methods. We should note that the architecture of student NIN without batch normalization is slightly different from teacher network, it doesn't have ReLU activations before pooling layers, which leads to better performance without batch normalization, and worse with. So to achieve the best performance with activation-based AT we had to train a new teacher, with batch normalization and without ReLU activations before pooling layers, and have AT losses on outputs of convolutional layers.

6.4.2 Large input image networks

In this section we experiment with hidden activation attention transfer on ImageNet networks which have 224×224 input image size. Presumably, attention matters more in this kind of networks as spatial resolution of attention maps is higher.

6.4.2.1 Transfer learning

To see how attention transfer works in finetuning we choose two datasets: Caltech-UCSD Birds-200-2011 fine-grained classification ("CUB") by Wah *et al.* (2011), and MIT indoor scene classification ("Scenes") by Quattoni and Torralba (2009), both containing around 5K images training images. We took ResNet-18 and ResNet-34 pretrained on ImageNet and finetuned on both datasets. On CUB we crop bounding boxes, rescale to 256 in one dimension and then take a random crop. Batch normalization layers are fixed for finetuning, and first group of residual blocks is frozen. We then took finetuned ResNet-34 networks and used them as teachers for ResNet-18 pretrained on ImageNet, with F^2_{sum} attention losses on 2 last groups. In both cases attention transfer provides significant improvements, closing the gap between ResNet-18 and ResNet-34 in accuracy. On Scenes AT works as well as KD, and on

type	model	ImageNet→CUB	ImageNet→Scenes
student	ResNet-18	28.5	28.2
KD	ResNet-18	27 (-1.5)	28.1 (-0.1)
AT	ResNet-18	27 (-1.5)	27.1 (-1.1)
teacher	ResNet-34	26.5	26

Table 6.4: Finetuning with attention transfer error on Scenes and CUB datasets

CUB AT works much better, which we speculate is due to importance of intermediate attention for fine-grained recognition. Moreover, after finetuning, student's attention maps indeed look more similar to teacher's (Fig. 6.6).

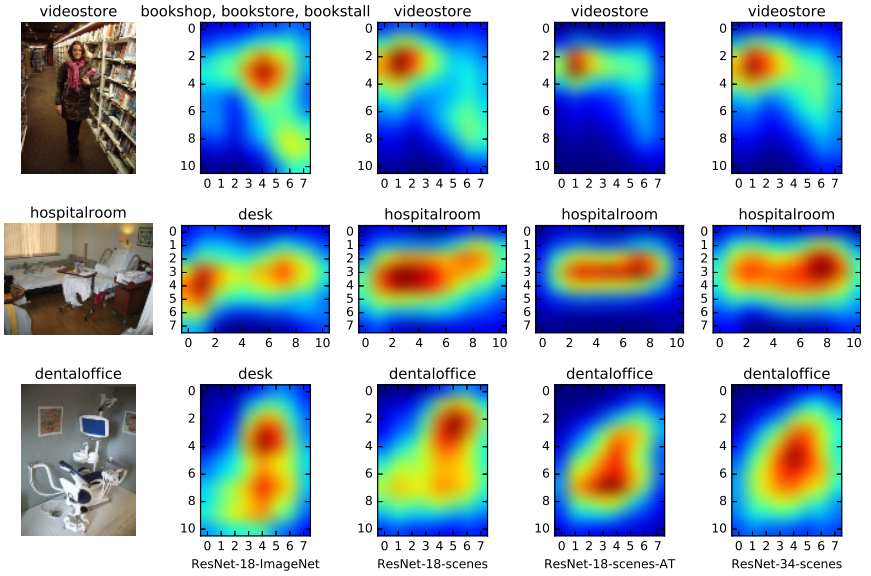
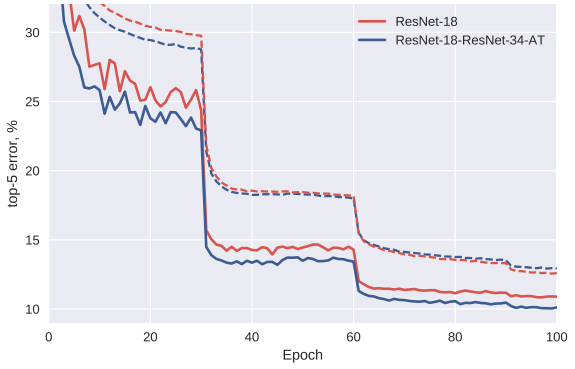


Figure 6.6: Top activation attention maps for different Scenes networks. Original pretrained ResNet-18 (ResNet-18-ImageNet), ResNet-18 trained on Scenes (ResNet-18-scenes), ResNet-18 trained with attention transfer (ResNet-18-scenes-AT) with ResNet-34 as a teacher, ResNet-34 trained on Scenes (ResNet-34-scenes). Predicted classes for each task are shown on top. Attention maps look more similar after transfer (images taken from test set).

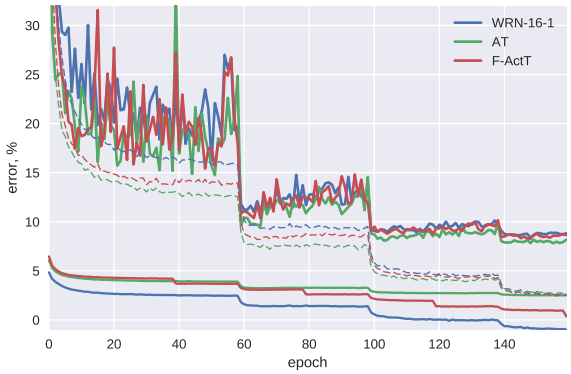
6.4.2.2 ImageNet

To showcase activation-based attention transfer on ImageNet we took ResNet-18 as a student, and ResNet-34 as a teacher, and tried to improve ResNet-18 accuracy. We added only two losses in the 2 last groups of residual blocks and used squared sum attention F^2_{sum}. ResNet-18 with attention transfer achieved 1.1% top-1 and 0.8% top-5 better validation accuracy (Table. 6.5 and Fig. 6.7a).

We were not able to achieve positive results with KD on ImageNet. With ResNet-18-ResNet-34 student-teacher pair it actually hurts convergence with the same hyperparameters as on CIFAR. As it was reported that KD struggles to work if teacher and student have different architecture/depth (we observe the same



(a) Attention transfer on ImageNet between ResNet-18 and ResNet-34. Solid lines represent top-5 validation error, dashed - top-5 training error. Two attention transfer losses were used on the outputs of two last groups of residual blocks respectively, no KD losses used.



(b) Activation attention transfer on CIFAR-10 from WRN-16-2 to WRN-16-1. Test error is in bold, train error is in dashed lines. Attention transfer greatly speeds up convergence and improves final accuracy.

Figure 6.7: Attention transfer convergence curves on CIFAR and ImageNet datasets

Model	top1, top5
ResNet-18	30.4, 10.8
AT	29.3, 10.0
ResNet-34	26.1, 8.3

Table 6.5: Attention transfer validation error (single crop) on ImageNet. Transfer losses are added on epoch 60/100.

on CIFAR), so we tried using the same architecture and depth for attention transfer. On CIFAR both AT and KD work well in this case and improve convergence and final accuracy, on ImageNet though KD converges significantly slower. We also could not find applications of FitNets, KD or similar methods on ImageNet in the literature. Given that, we can assume that proposed activation-based AT is the first knowledge transfer method to be successfully applied on ImageNet.

6.5 Conclusions

We presented several ways of transferring attention from one network to another, with experimental results over several image recognition datasets. It would be interesting to see how attention transfer works in cases where spatial information is more important, e.g. object detection or weakly-supervised localization.

Chapter 7

Discussion and future work

In this dissertation we showed that modern neural networks exhibit interesting novel properties, not yet observed in machine learning models. We only explored a few of them, and there is a lot of room for further exploration.

We started with patch matching in chapter 3, and showed that learned neural network descriptors can significantly outperform hand-crafted ones on this task. We also proposed an interesting way of comparing patches via a 2-channel network, which works better than siamese or triplet approaches. This finding could be interesting for other tasks that need to predict image or feature similarity. Since published, the work on patch comparing neural networks in Zagoruyko and Komodakis (2015) spanned a lot of research on not just learning better image descriptors, but the whole pipeline of detecting keypoints, computing descriptors, and matching altogether, which is still under active exploration.

In chapter 4 we presented a detailed study on object detection, and proposed novel MultiPathNet architecture in Zagoruyko *et al.* (2016), aggregating information via several paths before making final decision. This work was the first to propose streamlined instance segmentation and recognition approach with SharpMask and MultiPathNet systems, on a challenging COCO 2015 dataset. Despite ours, and more recent efforts, performance of neural networks on this dataset is far from human, and there is a lot work to be done in this direction.

Our work on wide residual networks in Zagoruyko and Komodakis (2016b), presented in chapter 5, switched the research on deep neural networks from focusing solely on depth to explore width as well, and spanned a lot of interesting research on network architectures. Also, it served as a baseline for countless number of papers on convolutional neural networks, due to simplicity and effectiveness of the proposed approach, and won several competitions in computer vision. It is still unclear why residual

connections are so effective, so we tried to address this question in our DiracNets work in Zagoruyko and Komodakis (2017a), in which we proposed an alternative simpler parameterization. We showed that DiracNets work as well ResNet on large datasets, but fall behind on smaller ones, which we plan to address in future. DiracNets are also interesting for theoretical analysis, due to their simplicity, and for understanding of trained networks.

Finally, in chapter 6 we presented a study on knowledge distillation, where we tried to transfer other information than network outputs, between teacher and student in Zagoruyko and Komodakis (2017b). Interestingly, we find that transferring full activations does not work as well as transferring attention, which we define as functions of activations or gradients. Attention transfer is not the only one way of doing knowledge distillation with intermediate features, there are other ways, which could potentially significantly improve training and final student performance, as well as understanding of what's important for neural networks.

We hope that these small steps will lead to improvements in our understanding of neural networks, and, ultimately, to more intelligent systems.

Bibliography

Alexe, B., Deselaers, T., and Ferrari, V. (2012). Measuring the objectness of image windows. *PAMI*.

Arbeláez, P., Pont-Tuset, J., Barron, J., Marques, F., and Malik, J. (2014). Multiscale combinatorial grouping. In *CVPR*.

Ba, J., Kiros, R., and Hinton, G. E. (2016). Layer normalization. *CoRR*, **abs/1607.06450**.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *CoRR*, **abs/1409.0473**.

Balntas, V., Johns, E., Tang, L., and Mikolajczyk, K. (2016). Pn-net: Conjoined triple deep network for learning local image descriptors. *CoRR*, **abs/1601.05030**.

Bay, H., Tuytelaars, T., and Gool, L. V. (2006). Surf: Speeded up robust features. In *In ECCV*, pages 404–417.

Becker, S. and LeCun, Y. (1989). Improving the convergence of back-propagation learning with second-order methods. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proc. of the 1988 Connectionist Models Summer School*, pages 29–37, San Mateo. Morgan Kaufman.

Bell, S., Zitnick, C. L., Bala, K., and Girshick, R. (2016). Inside-outside net: Detecting objects in context with skip pooling and recurrent neural nets. In *CVPR*.

Bengio, Y. and Glorot, X. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of AISTATS 2010*, volume 9, pages 249–256.

Bengio, Y. and LeCun, Y. (2007). Scaling learning algorithms towards AI. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large Scale Kernel Machines*. MIT Press.

Bertinetto, L., Valmadre, J., Henriques, J. F., Vedaldi, A., and Torr, P. H. (2016). Fully-convolutional siamese networks for object tracking. In *European Conference on Computer Vision*, pages 850–865. Springer.

Bianchini, M. and Scarselli, F. (2014). On the complexity of shallow and deep neural network classifiers. In *22th European Symposium on Artificial Neural Networks, ESANN 2014, Bruges, Belgium, April 23-25, 2014*.

Boix, X., Gygli, M., Roig, G., and Van Gool, L. (2013). Sparse quantization for patch description. In *CVPR*.

Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., and Shah, R. (1993). Signature verification using a siamese time delay neural network. In *NIPS*.

Brown, M., Hua, G., and Winder, S. (2011). Discriminative learning of local image descriptors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, **33**(1), 43–57.

Bryson, A. E. (1961). A gradient method for optimizing multi-stage allocation processes. In *Proc. Harvard Univ. Symposium on digital computers and their applications*.

Bucila, C., Caruana, R., and Niculescu-Mizil, A. (2006). Model compression. In *KDD*, pages 535–541.

Calonder, M., Lepetit, V., and Fua, P. (2010). Brief: Binary robust independent elementary features.

Chatfield, K., Simonyan, K., Vedaldi, A., and Zisserman, A. (2014). Return of the devil in the details: Delving deep into convolutional nets. In *British Machine Vision Conference*.

Chen, T., Goodfellow, I., and Shlens, J. (2016). Net2net: Accelerating learning via knowledge transfer. In *International Conference on Learning Representation*.

Chopra, S., Hadsell, R., and LeCun, Y. (2005). Learning a similarity metric discriminatively, with application to face verification. In *CVPR*.

Choy, C. B., Gwak, J., Savarese, S., and Chandraker, M. (2016). Universal correspondence network. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2414–2422. Curran Associates, Inc.

Clevert, D., Unterthiner, T., and Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). *CoRR*, **abs/1511.07289**.

Cohen, T. and Welling, M. (2016). Group equivariant convolutional networks. In *ICML*.

Collobert, R., Kavukcuoglu, K., and Farabet, C. (2011). Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*.

Conejo, B., Komodakis, N., Leprince, S., and Avouac, J.-P. (2014). Inference by learning: Speeding-up graphical model optimization via a coarse-to-fine cascade of pruning classifier. In *NIPS*.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, **2**(4), 303–314.

Dai, J., He, K., and Sun, J. (2016). Instance-aware semantic segmentation via multi-task network cascades. In *CVPR*.

Deng, J., Dong, W., Socher, R., Li, L., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *CVPR*.

Denil, M., Bazzani, L., Larochelle, H., and de Freitas, N. (2012). Learning where to attend with deep architectures for image tracking. *Neural Computation*.

Dollár, P., Appel, R., Belongie, S., and Perona, P. (2014). Fast feature pyramids for object detection. *PAMI*.

Dreyfus, S. E. (1973). The computational solution of optimal control problems with time lag. *IEEE Transactions on Automatic Control*, **18(4)**, 383–385.

Drucker, H. and LeCun, Y. (1992). Improving generalization performance using double backpropagation. *IEEE Transaction on Neural Networks*, **3**(6), 991–997.

Duchi, J. C., Hazan, E., and Singer, Y. (2010). Adaptive subgradient methods for online learning and stochastic optimization. In *COLT*.

Dumoulin, V. and Visin, F. (2016). A guide to convolution arithmetic for deep learning. *ArXiv e-prints*.

Eigen, D., Puhrsch, C., and Fergus, R. (2014). Depth map prediction from a single image using a multi-scale deep network. In *NIPS*.

Everingham, M., Gool, L. V., Williams, C. K. I., Winn, J., and Zisserman, A. (2010). The PASCAL visual object classes (VOC) challenge. *IJCV*.

Faugeras, O., Viéville, T., Theron, E., Vuillemin, J., Hotz, B., Zhang, Z., Moll, L., Bertin, P., Mathieu, H., Fua, P., Berry, G., and Proy, C. (1993). Real-time correlation-based stereo : algorithm, implementations and applications. Research Report RR-2013, INRIA.

Felzenszwalb, P. F., Girshick, R. B., McAllester, D., and Ramanan, D. (2010). Object detection with discriminatively trained part-based models. *PAMI*.

Fischer, P., Dosovitskiy, A., and Brox, T. (2014). Descriptor matching with convolutional neural networks: a comparison to SIFT. *CoRR*, **abs/1405.5769**.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, **36**, 193–202.

G, V. K. B., Carneiro, G., and Reid, I. D. (2016). Learning local image descriptors with deep siamese and triplet convolutional networks by minimizing global loss functions. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5385–5394.

Gauss, C. F. (1809). *Theoria motus corporum coelestium in sectionibus conicis solem ambientium*.

Gidaris, S. and Komodakis, N. (2015). Object detection via a multi-region and semantic segmentation-aware cnn model. In *ICCV*.

Gidaris, S. and Komodakis, N. (2016). Locnet: Improving localization accuracy for object detection. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*.

Girshick, R. (2015). Fast R-CNN. In *ICCV*.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*.

Goesele, M., Snavely, N., Curless, B., Hoppe, H., and Seitz, S. M. (2007). Multi-view stereo for community photo collections. In *Proceedings of the 11th International Conference on Computer Vision (ICCV 2007)*, pages 265–270, Rio de Janeiro, Brazil. IEEE.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013). Maxout networks. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning (ICML'13)*, pages 1319–1327.

Graham, B. (2014). Fractional max-pooling. *arXiv:1412.6071*.

Han, X., Leung, T., Jia, Y., Sukthankar, R., and Berg, A. C. (2015). Matchnet: Unifying feature and metric learning for patch-based matching. In *CVPR*.

Hariharan, B., Arbeláez, P., Girshick, R., and Malik, J. (2015). Hypercolumns for object segmentation and fine-grained localization. In *CVPR*.

He, K., Zhang, X., Ren, S., and Sun, J. (2014). Spatial pyramid pooling in deep convolutional networks for visual recognition. In *ECCV*.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034.

He, K., Zhang, X., Ren, S., and Sun, J. (2016a). Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

He, K., Zhang, X., Ren, S., and Sun, J. (2016b). Identity mappings in deep residual networks. In *ECCV*.

Hinton, G. E., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural networks.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory.

Hoffer, E. and Ailon, N. (2015). Deep metric learning using triplet network. In *SIMBAD*.

Hoiem, D., Chodpathumwan, Y., and Dai, Q. (2012). Diagnosing error in object detectors. In *ICCV*.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings National Academy of Science*, **79**, 2554–2558.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, **2**(5), 359–366.

Hosang, J., Benenson, R., Dollár, P., and Schiele, B. (2015). What makes for effective detection proposals? *PAMI*.

Huang, G., Sun, Y., Liu, Z., Sedra, D., and Weinberger, K. Q. (2016). Deep networks with stochastic depth. In *ECCV*.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In D. Blei and F. Bach, editors, *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 448–456. JMLR Workshop and Conference Proceedings.

Ivakhnenko, A. G. (1968). The group method of data handling – a rival of the method of stochastic approximation. *Soviet Automatic Control*, **13**(3), 43–55.

Ivakhnenko, A. G. (1971). Polynomial theory of complex systems. *IEEE Transactions on Systems, Man and Cybernetics*, (4), 364–378.

Ivakhnenko, A. G. and Lapa, V. G. (1965). *Cybernetic Predicting Devices*. CCM Information Corporation.

Ivakhnenko, A. G., Lapa, V. G., and McDonough, R. N. (1967). *Cybernetics and forecasting techniques*. American Elsevier, NY.

Kelley, H. J. (1960). Gradient theory of optimal flight paths. *ARS Journal*, **30**(10), 947–954.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, **abs/1412.6980**.

Kohonen, T. (1972). Correlation matrix memories. *Computers, IEEE Transactions on*, **100**(4), 353–359.

Kohonen, T. (1988). *Self-Organization and Associative Memory*. Springer, second edition.

Komodakis, N., Tziritas, G., and Paragios, N. (2007). Fast, approximately optimal solutions for single and dynamic MRFs. In *CVPR*.

Kozinski, M., Gadde, R., Zagoruyko, S., Obozinski, G., and Marlet, R. (2015). A mrf shape prior for facade parsing with occlusions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2820–2828.

Krizhevsky, A. (2014). One weird trick for parallelizing convolutional neural networks. *arXiv:1404.5997*.

Krizhevsky, A., Nair, V., and Hinton, G. (2012a). Cifar-10 (canadian institute for advanced research).

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012b). Imagenet classification with deep convolutional neural networks. In *NIPS*.

Larochelle, H. and Hinton, G. E. (2010). Learning to combine foveal glimpses with a third-order boltzmann machine. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 1243–1251. Curran Associates, Inc.

Larochelle, H., Erhan, D., Courville, A., Bergstra, J., and Bengio, Y. (2007). An empirical evaluation of deep architectures on problems with many factors of variation. In Z. Ghahramani, editor, *Proceedings of the 24th International Conference on Machine Learning (ICML'07)*, pages 473–480. ACM.

Lecun, Y. (1987). *PhD thesis: Modeles connexionnistes de l'apprentissage (connectionist learning models)*. Universite P. et M. Curie (Paris 6).

LeCun, Y. (1988). A theoretical framework for back-propagation. In *Proceedings of the 1988 Connectionist Models Summer School*, pages 21–28.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*.

Lee, C.-Y., Xie, S., Gallagher, P., Zhang, Z., and Tu, Z. (2014). Deeply-Supervised Nets.

Legendre, A. M. (1805). *Nouvelles méthodes pour la détermination des orbites des cometes*. F. Didot.

Lei, J. B. and Caruana, R. (2014). Do deep nets really need to be deep? In *NIPS*.

Leutenegger, S., Chli, M., and Siegwart, Y. (2011). Brisk: Binary robust invariant scalable keypoints. In *In Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2548–2555.

Lin, M., Chen, Q., and Yan, S. (2013). Network in network. *CoRR*, **abs/1312.4400**.

Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., and Dollár, P. (2015). Microsoft COCO: Common objects in context. *arXiv:1405.0312*.

Linnainmaa, S. (1970). *The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors*. Master's thesis, Univ. Helsinki.

Linnainmaa, S. (1976). Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*, **16**(2), 146–160.

Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *CVPR*.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, **60**, 91–110.

Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning (ICML)*.

Martens, J. and Grosse, R. B. (2015). Optimizing neural networks with kronecker-factored approximate curvature. In *ICML*.

McCulloch, W. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, **7**, 115–133.

Mikolajczyk, K. and Schmid, C. (2005). A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, **27**(10), 1615–1630.

Mnih, V., Heess, N., Graves, A., and Kavukcuoglu, K. (2014). Recurrent models of visual attention. In *NIPS*.

Montúfar, G. F., Pascanu, R., Cho, K., and Bengio, Y. (2014). On the number of linear regions of deep neural networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2924–2932.

Nowak, E. and Jurie, F. (2007). Learning Visual Similarity Measures for Comparing Never Seen Objects. In *CPVR 2007 - IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, Minneapolis, United States. IEEE Computer society.

Oquab, M., Bottou, L., Laptev, I., and Sivic, J. (2015). Is object localization for free? – weakly-supervised learning with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Oyallon, E., Belilovsky, E., and Zagoruyko, S. (2017). Scaling the scattering transform: Deep hybrid networks. In *The IEEE International Conference on Computer Vision (ICCV)*.

Papadomanolaki, M., Vakalopoulou, M., Zagoruyko, S., and Karantzalos, K. (2016). Benchmarking deep learning frameworks for the classification of very high resolution satellite multispectral data. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, **III-7**, 83–88.

Parker, D. B. (1985). Learning-logic. Technical Report TR-47, Center for Comp. Research in Economics and Management Sci., MIT.

Parkhi, O. M., Vedaldi, A., and Zisserman, A. (2015). Deep face recognition. In *British Machine Vision Conference*.

Pinheiro, P. O., Collobert, R., and Dollár, P. (2015). Learning to segment object candidates. In *NIPS*.

Pinheiro, P. O., Lin, T.-Y., Collobert, R., and Dollár, P. (2016). Learning to refine object segments. In *ECCV*.

Polyak, B. (1964). Some methods of speeding up the convergence of iteration methods. In *Ussr Computational Mathematics and Mathematical Physics*, volume 4, pages 1–17.

Pontryagin, L. S., Boltyanskii, V. G., Gamrelidze, R. V., and Mishchenko, E. F. (1961). *The Mathematical Theory of Optimal Processes*.

Quattoni, A. and Torralba, A. (2009). Recognizing indoor scenes. In *CVPR*.

Raiko, T., Valpola, H., and Lecun, Y. (2012). Deep learning made easier by linear transformations in perceptrons. In N. D. Lawrence and M. A. Girolami, editors, *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics (AISTATS-12)*, volume 22, pages 924–932.

Razavian, A. S., Azizpour, H., Sullivan, J., and Carlsson, S. (2014). CNN features off-the-shelf: An astounding baseline for recognition. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2014, Columbus, OH, USA, June 23-28, 2014*, pages 512–519.

Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*.

Rensink, R. A. (2000). The dynamic representation of scenes. In *Visual Cognition*, pages 17–42.

Romero, A., Ballas, N., Ebrahimi Kahou, S., Chassang, A., Gatta, C., and Bengio, Y. (2014). FitNets: Hints for thin deep nets. Technical Report Arxiv report 1412.6550, arXiv.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, **65**(6), 386.

Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011). Orb: An efficient alternative to sift or surf. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pages 2564–2571, Washington, DC, USA. IEEE Computer Society.

Salimans, T. and Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Neural Information Processing Systems 2016*.

Schmidhuber, J. (1992). Learning complex, extended sequences using the principle of history compression. *Neural Computation*, **4**(2), 234–242.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks : the official journal of the International Neural Network Society*, **61**, 85–117.

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Batra, D. (2017). Grad-cam: Visual explanations from deep networks via gradient-based localization. In *The IEEE International Conference on Computer Vision (ICCV)*.

Sermanet, P., Kavukcuoglu, K., Chintala, S., and LeCun, Y. (2013). Pedestrian detection with unsupervised multi-stage feature learning. In *CVPR*.

Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., and LeCun, Y. (2014). Overfeat: Integrated recognition, localization and detection using convolutional networks. In *ICLR*.

Simo-Serra, E., Trulls, E., Ferraz, L., Kokkinos, I., Fua, P., and Moreno-Noguer, F. (2015). Discriminative Learning of Deep Convolutional Feature Point Descriptors. In *Proceedings of the International Conference on Computer Vision (ICCV)*.

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *ICLR*.

Simonyan, K., Vedaldi, A., and Zisserman, A. (2014). Deep inside convolutional networks: Visualising image classification models and saliency maps. In *ICLR Workshop*.

Snavely, N., Seitz, S. M., and Szeliski, R. (2006). Photo tourism: Exploring photo collections in 3d. *ACM Trans. Graph.*, **25**(3), 835–846.

Snavely, N., Seitz, S. M., and Szeliski, R. (2008). Modeling the world from internet photo collections. *Int. J. Comput. Vision*, **80**(2), 189–210.

Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. (2015). Striving for simplicity: The all convolutional net. In *arXiv:1412.6806, also appeared at ICLR 2015 Workshop Track*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *JMLR*.

Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015). Training very deep networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2377–2385. Curran Associates, Inc.

Strecha, C., von Hansen, W., Gool, L. J. V., Fua, P., and Thoennessen, U. (2008). On benchmarking camera calibration and multi-view stereo for high resolution imagery. In *CVPR*. IEEE Computer Society.

Sutskever, I., Martens, J., Dahl, G. E., and Hinton, G. E. (2013). On the importance of initialization and momentum in deep learning. In S. Dasgupta and D. Mcallester, editors, *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 1139–1147. JMLR Workshop and Conference Proceedings.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. J., and Fergus, R. (2013). Intriguing properties of neural networks. *CoRR*, **abs/1312.6199**.

Szegedy, C., Reed, S., Erhan, D., and Anguelov, D. (2014). Scalable, high-quality object detection. *arXiv:1412.1441*.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *CVPR*.

Szegedy, C., Ioffe, S., and Vanhoucke, V. (2016). Inception-v4, inception-resnet and the impact of residual connections on learning.

Taigman, Y., Yang, M., Ranzato, M., and Wolf, L. (2014). Deepface: Closing the gap to human-level performance in face verification. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Tao, R., Gavves, E., and Smeulders, A. W. (2016). Siamese instance search for tracking. *arXiv preprint arXiv:1605.05863*.

Tola, E., V.Lepetit, and Fua, P. (2008). A Fast Local Descriptor for Dense Matching. In *Proceedings of Computer Vision and Pattern Recognition*, Alaska, USA.

Torralba, A. (2003). Contextual priming for object detection. *IJCV*.

Trzcinski, T., Christoudias, C. M., Lepetit, V., and Fua, P. (2012). Learning image descriptors with the boosting-trick. In *NIPS*.

Trzcinski, T., Christoudias, C. M., Fua, P., and Lepetit, V. (2013). Boosting binary keypoint descriptors. *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2874–2881.

Uijlings, J., van de Sande, K., Gevers, T., and Smeulders, A. (2013). Selective search for object recog. *IJCV*.

Vasilache, N., Johnson, J., Mathieu, M., Chintala, S., Piantino, S., and LeCun, Y. (2014). Fast convolutional nets with fbfft: A GPU performance evaluation. *CoRR*, **abs/1412.7580**.

Viola, P. and Jones, M. J. (2004). Robust real-time face detection. *IJCV*.

Wah, C., Branson, S., Welinder, P., Perona, P., and Belongie, S. (2011). The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology.

Werbos, P. J. (1981). Applications of advances in nonlinear sensitivity analysis. In *Proceedings of the 10th IFIP Conference, 31.8 - 4.9, NYC*, pages 762–770.

Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., and Recht, B. (2017). The marginal value of adaptive gradient methods in machine learning. In *NIPS*.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A. C., Salakhutdinov, R., Zemel, R. S., and Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. In *ICML*.

Yang, Z., He, X., Gao, J., Deng, L., and Smola, A. J. (2016). Stacked attention networks for image question answering. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21–29.

Yi, K. M., Verdie, Y., Fua, P., and Lepetit, V. (2016a). Learning to assign orientations to feature points. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 107–116.

Yi, K. M., Trulls, E., Lepetit, V., and Fua, P. (2016b). Lift: Learned invariant feature transform. In *ECCV*.

Zagoruyko, S. and Komodakis, N. (2015). Learning to compare image patches via convolutional neural networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zagoruyko, S. and Komodakis, N. (2016a). Deep compare: A study on using convolutional neural networks to compare image patches. *Computer Vision and Image Understanding Special Issue: Deep Learning*.

Zagoruyko, S. and Komodakis, N. (2016b). Wide residual networks. In *BMVC*.

Zagoruyko, S. and Komodakis, N. (2017a). Diracnets: Training very deep neural networks without skip-connections.

Zagoruyko, S. and Komodakis, N. (2017b). Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. In *ICLR*.

Zagoruyko, S., Lerer, A., Lin, T.-Y., Pinheiro, P. O., Gross, S., Chintala, S., and Dollár, P. (2016). A multipath network for object detection. In *BMVC*.

Zbontar, J. and LeCun, Y. (2015). Computing the stereo matching cost with a convolutional neural network. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1592–1599.

Zeiler, M. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *ECCV*.

Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., and Torralba, A. (2016). Learning deep features for discriminative localization. In *Computer Vision and Pattern Recognition*.

Zitnick, C. L. and Dollár, P. (2014). Edge boxes: Locating object proposals from edges. In *ECCV*.