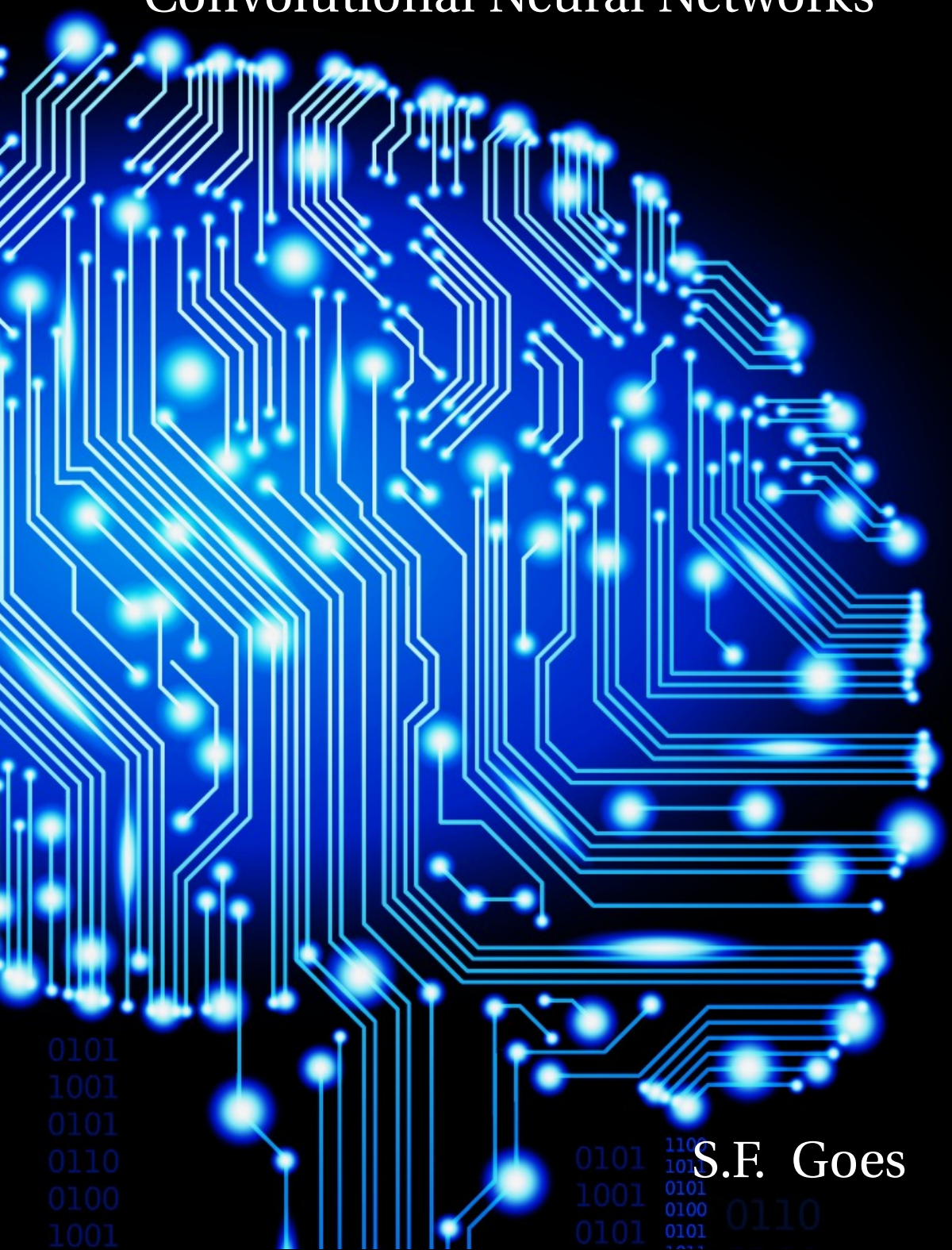
Learning the scale of image features in Convolutional Neural Networks



S.F. Goes

LEARNING THE SCALE OF IMAGE FEATURES IN CONVOLUTIONAL NEURAL NETWORKS

Afstudeerscriptie

ter verkrijging van de graad van Master of Science in de Technische Natuurkunde
aan de Technische Universiteit Delft,
in het openbaar te verdedigen op woensdag 4 oktober 2017 om 10:00 uur

door

Sten GOES

Bachelor of Science in de Technische Natuurkunde
Technische Universiteit Delft, Delft, Nederland,
geboren te Woerden, Nederland.

Dit afstudeerverslag is goedgekeurd door de

promotor: prof. dr. ir. L.J. van Vliet

Samenstelling afstudeercommissie:

Begeleiders:	
Prof. dr. ir. L.J. van Vliet	Quantitative Imaging
Dr. J.C. van Gemert,	Pattern Recognition & Bioinformatics (EEMCS)
Onafhankelijke leden:	
Dr. F.M. Vos	Quantitative Imaging
Dr. ir. D.J. Verschuur	Acoustical Wavefield Imaging

Dr. J.C. van Gemert heeft in belangrijke mate aan de totstandkoming van deze afstudeerscriptie bijgedragen.

Keywords: machine learning, deep learning, convolutional neural networks, scale-space, learning feature scale

Printed by: www.printenbind.nl

Front & Back: cover art that captures the entire content of this thesis in a single illustration.

An electronic version of this thesis is available at
http://repository.tudelft.nl/.

Success is not final, failure is not fatal: it is the courage to continue that counts.

Winston Churchill

CONTENTS

PREFACE

About a year ago, I was looking for a research subject for my Master Thesis. I really wanted to do something with Machine Learning and Computer Vision. So, the pattern recognition group at the faculty of EEMCS was a natural place to start looking. I remember that Jan proposed a couple of other possibilities, before arriving at the subject of learning the scale of image features in convolutional neural networks. I must admit that I did not immediately see why this would be important, but I liked the discussions we had and I thought Jan would make a good supervisor. Officially however, I needed a promotor from the faculty of applied sciences. It was therefore great to hear that Lucas was also enthusiastic about the project, and that he wanted to become my promoter despite his busy schedule. About a month later, I started the project and it has been a wonderful and educational journey ever since.

Right from the start, I decided to containerize my code inside a light-weight Linux virtual machine with Docker. This turned out to be good decision, because I have ran most of my experiments in the cloud on high-end GPU servers inside Amazon, Microsoft or Google data centers. By using docker containers, I could get my experiments up and running on any machine in just a couple of seconds. This has taught me a lot about Docker, Linux and Cloud computing. I think all three are valuable skills for the future.

In the beginning progress went quite fast, and we quickly obtained our first results on simple and small convolutional neural networks. The results looked quite alright, but it took quite some time to fully understand all the delicate intricacies before we could move on to more complex neural networks. This is were some of the trouble started. It turned out the filter sizes became quite large during the training stage and that therefore the training time was much longer than for normal neural networks. It took me several months to re-implement the convolutional layer in the frequency domain and to speed up the performance. Fortunately, I was not on my own in my quest to decrease training times of neural networks. In the meantime: Amazon had upgraded their servers with much faster GPUs, Google had released 3 newer versions of the Tensorflow framework, and Nvidia had released a newer version of their CuDNN library (a library with GPU primitives for neural networks on which almost all deep learning frameworks are built). All these improvements combined, resulted in a very large overall improvement which made it possible to train complex networks in a couple of days rather than a couple of weeks. It is therefore only very recent that all the interesting results came in, and filled in the empty pieces of the puzzle.

With this work, I hope that I have created a solid foundation on which other students can continue to build. The code has been written in C++ as custom Tensorflow operation and can be imported as self-contained module in Python. In principle this module can

be used in experiments without any modification. However if modification is needed, one does not have to worry about checking gradients or verifying thousands of lines of code. The unit tests will immediately reveal when the code has been broken. Furthermore, the docker image should make it super easy to reproduce or extend the work that I have done. Everyone should be able to get an experiment up and running on his or her machine with just a single command.

I hope you will enjoy reading this thesis.

Sten Goes
Delft, September 2017

ACKNOWLEDGEMENTS

The first words of this thesis will be devoted to thanking the people who helped me reach this point. First of all, I want to express my gratitude to Lucas and Jan. I was very lucky to have you both as my supervisors. Thank you Lucas for always showing interest in my work despite your busy schedule and thank you for the interesting discussions we had during our meetings. These discussions made clear to me, that you know a lot about a lot of things and that you care passionately about doing research. Something what I as a physicist, can only admire. Thank you Jan, for your everlasting enthusiasm and all your encouragements. You let me be the manager of my own project, while pointing me in the right direction whenever I lost track. Your comments are always very valuable and your optimism is very inspiring! Partially because of you, the project became one of the highlights of my study.

I would like to express my deepest appreciation for my friends from Jungle AI. Tim, Silvio and Alexander, I really cannot thank you guys enough for providing me with access to awesome GPU servers on the Microsoft cloud platform. It is definitely not a coincidence that the all the interesting results came in, after you started to support me. It therefore goes without saying, that this thesis would have looked very differently without your help. It was a pleasure to join you guys for one day at your office and I am looking forward to the day that I can do something back.

I also want to thank all the people from the Quantitive Imaging section for all the interesting discussions and the pleasant time during my stay. A special thanks goes to Annelies for planning the date and location of my defense.

De allerlaatste woorden van dit dankwoord zijn bestemd voor de mensen die in het niet-academische aspect van mijn leven belangrijk voor mij zijn. Mijn huisgenoten, bedankt voor al die keren dat jullie voor me hebben gekookt als ik weer eens tot laat door bleef werken. Sorry voor al die keren dat ik onophoudelijk door bleef praten over waarom machine learning zo gaaf is en waarom het ons leven gaat veranderen. Setiawan en Jenny, bedankt voor jullie gastvrijheid. De open armen waarmee ik altijd bij jullie word ontvangen, is zeer bewonderingswaardig. Pap en Mam, bedankt voor jullie onvoorwaardelijke steun, altijd en overal. Mam, bedankt voor al je goede zorgen. Pap, jij bent zonder twijfel de persoon waar ik het meest tegenop kijk. Al zo lang als ik me kan herinneren, werk je altijd hard en maak je lange dagen. De tomeloze inzet waarmee jij je al tientallen jaren inzet voor de Florigo en daarmee voor onze hele familie, maakt mij erg trots! En tenslotte Nancy, dankjewel voor al je geduld en begrip. In slechte tijden ben je er altijd om me te steunen, en in betere tijden ben je er om samen de succesen te vieren. Zelfs in de donkerste tijden, geef jij mijn leven kleur. Op naar mooie avonturen samen in Amerika!

SUMMARY

The millions of filter weights in Convolutional Neural Networks (CNN), all have a well-defined and analytical expression for the partial derivative to the loss function. Hence, they can be learned with gradient based optimization. Whereas the filter weights have a well-defined and analytical derivative, the filter size has not. The main reason for this, is that the filter size is a discretely valued parameter instead of a continuously valued one. The filter size is thus a fixed parameter, that needs to be selected before the optimization (training) of the filter weights starts. There is currently no other way to optimize filter sizes, than with an exhaustive search over multiple CNNs trained with with different filter sizes.

In this report we propose a new filter called *Structured Receptive Field* of which the filter size can be optimized during the training stage. This new filter parameterizes a normal filter as a linear combination of all 2D gaussian derivatives up to a certain order. Instead of learning the weights of the resulting filter directly, we learn the weights of the linear combination and the sigma of the gaussian derivatives that implicitly define the filter weights. Furthermore, instead of having a fixed size, the new filter has a variable size which is determined by the value of the sigma. We truncate the filter size at 3 sigmas from the center (rounded upwards to the nearest integer), because at that point the gaussian derivatives already contain ~99.7% of their total volume and the border values are almost zero. The advantage of parameterizing normal filters in this way, is that gradient descent optimization of the continuous sigma parameter which has a well-defined derivative, can be used as a proxy for optimizing the discrete filter size which has no well-defined derivative.

The basic idea for this parameterization comes from scale-space theory in which the scale of image features is studied by parameterizing the features with a continuous scale parameter. Thereby explicitly decoupling the spatial structure of image feature from the scale at which it occurs in the image. Structured receptive fields have several compelling advantages: they can learn bigger filters without increasing the number of learnable parameters and without increasing the complexity of the structure of the filter, their structure is solely and explicitly controlled by the truncation order of the gaussian derivatives and the value of the learned continuous scale parameter can provide valuable insights into the workings and the importance of subsampling layers in CNN architectures.

In this report we both provide theoretical and empirical evidence that structured receptive fields can approximate any filter learned by normal CNNs. I.e. we show that structured receptive fields of order 4 can approximate the filters in all layers of a pre-trained AlexNet. Furthermore we demonstrate empirically that structured receptive fields can indeed learn their own filter size during training, and that this extra ability over normal

filters seems to give them an advantage over normal filters when used for classification tasks. I.e. by replacing normal filters in a DenseNet architecture and keeping everything else the same, a ~1% higher test accuracy was obtained on the highly competitive CIFAR-10 benchmark dataset. It would not be wise to draw hard conclusions from a single training run on a single dataset, but this result definitely looks promising. In future research we hope to demonstrate that, because of their extra ability to learn their filter size, replacing normal filters with structured receptive fields will always lead to strictly better or equal performance.

This thesis was written in partial fulfillment of the requirements for the degree of Master of Science in Applied Physics at Delft University of Technology. The research has been conducted in the section quantitive imaging at the faculty of Applied Sciences, and has been performed in close collaboration with the section pattern recognition and bioinformatics at the faculty of Electrical Engineering, Mathematics and Computer Science.

After installing docker, a pre-configured virtual machine can be started with one single command:
`docker run -it -p 8888:8888 -p 6006:6006 stengoes/structured-receptive-fields:1.0.0-cpu`

The code used in this research is also available at Github:
`https://github.com/stengoes/structured-receptive-fields`

1

INTRODUCTION

1.1. BACKGROUND

The research field of *Artificial Intelligence* (AI) studies how machines can be made intelligent. A machine is said to be intelligent, if it can perform one or multiple tasks at a performance level that is comparable or better than that of humans [1]. *Computer vision* is a subfield of AI, that focusses solely on building AI systems that solve visual tasks. Some of the most fundamental visual tasks are classification (what is seen?), localization (where is it seen?) and segmentation (which pixels belong to which objects?) [2].

For many years, researchers tried to solve these tasks by using a model driven approach. They tried to model the algorithm for recognizing objects with a set of hand-coded rules. For example, they would first use hand-coded algorithms to detected edges, ridges, corners and blobs [3][4][5][6]. From there they would build more sophisticated rules and algorithms to classify or localize the objects of interest. Some of these rule-based systems actually worked quite well in specifically adjusted environments such as industrial settings, where illumination can be kept constant, partial occlusion and background clutter can be avoided, and where objects usually have fixed shapes (no elastic deformations). However, it turned out to be very hard to come up with models that would make these algorithms work in 'the wild' where you do have to deal with problems such as illumination changes, elastic deformations and intraclass variation.

The big successes came when researchers started to abandon the model driven approach and started to use a data driven approach instead. Using a data driven approach to build an AI system is called *Machine Learning*. The idea is that you have a large dataset of example images of which the correct results are annotated by a human expert. The hopes are that a model with a low prediction error on this annotated dataset, will also perform well on new and unseen examples that are not annotated [7, 8].

The concepts of machine learning and neural networks are explained in more detail in chapter 2 and 3.

1

Machine learning algorithms are basically mathematical functions that contain free parameters and the best values for these parameters can be found by minimizing a loss function over the dataset of annotated examples. The data driven approach therefore turns the problem of designing a model into a minimization problem. Hence, we no longer have to worry about how to design our model, but we let the data do that for us. Machine learning is particularly useful in situations where it is hard to come up with a model, but it is easy to collect a large dataset of annotated examples [7, 8].

Since the early beginning of AI, researchers have tried to make machine learning work for computer vision. Only recently however, large datasets [9] have become available (Big data) and computers have become powerful enough (GPU programming) [10, 11] to train the very complex models that are needed to solve these difficult tasks.

For supervised machine learning, it is common to make a distinction between regression problems and classification problems. In regression problems the goal is to predict the outcome of a continuously valued function, whereas in classification problems the goal is to identify group membership. For example, predicting a persons age based on an image is a regression problem and identifying an object in an image as an apple, an orange or a banana is a classification problem. Note that regression and classification problems are actually very similar. A classification problem can be formulated as a regression problem of a decision boundary, with the additional step of checking on which side of the decision boundary an example lies. Most machine algorithms that can solve regression problems can thus be extended to solve classification problems as well [7, 12].

There are many different machine learning algorithms that all have slightly different parametric formulas. Some examples are: linear and logistic regression [13, 14], regression and decision trees [15], support vector machines [16], bayesian algorithms, k-nearest neighbor algorithms [17] and *Artificial Neural Networks* [18, 19].

Of all those different algorithms, artificial neural networks are particularly powerful because they can learn the complex relationships that are needed to solve complex tasks. Whereas the other machine learning algorithms make prior assumptions on the mathematical form of the relationship between input and output variables (i.e. linear regression assumes a linear relationship), neural networks with a sufficient amount of neurons can approximate any relationship without making these assumptions [20–22].

Artificial neural networks are biologically inspired by the human brain. Neuroscientists discovered that certain brain parts can take over each others functions [23–25]. However this does not happen automatically. Instead, when taking over functions these parts need re-learn these functions. This caused some AI researchers and neuroscientists to believe that the different brain parts can be modeled by exactly the same algorithm and that this algorithm can be tuned for a particular task by changing the values of its parameters [26] [1].

[1] This is sometimes called the 'one program' hypothesis and is supported by neuroscientist Jeff Hawkins [26] and AI researcher Andrew Ng.

A *Convolutional Neural Network* (CNN) is a type of neural network that is specifically designed to solve tasks involving image data. Whereas other machine learning algorithms suffer from the *curse of dimensionality* and cannot handle the high dimensional nature of raw images (e.g. a separate dimension for each pixel), CNNs are specifically designed to handle raw images directly [19]. A CNN can therefore optimize the whole pipeline from the input image to the desired output, which makes it a powerful algorithm [11].

A CNN gradually transforms the input image into the desired output via a series of mathematical operations of which convolution is the most important one. These mathematical operations are commonly referred to as layers and they are stacked in a feedforward way. Meaning that the output of the previous layer is the input of the following layer.

1.2. PROBLEM DESCRIPTION

The convolutional layer is the core building block of CNNs. It takes a multi-channel image as input and convolves it with a set of filters, each filter produces a one-channel output image called a feature map. Convolving an image with a filter, means sliding the filter (a small window: i.e. 5x5 pixels) over the image and computing a dot product between the underlying image and the filter weights at every location. High values of the dot product in the feature map, indicate that input image is very similar to the filter at that location. The feature maps of different filters indicate the presence and absence of different image features (small patches), at spatial locations in the input image [12, 27].

The next convolutional layer can treat this stack of feature maps as a new multi-channel input image to detect the presence or absence of more high level features. By stacking multiple convolutional layers on top of each other, detection of high level features such a faces can be decomposed in detection of lower level features such as mouths, eyes, noses and ears. The detection of these features can then be decomposed in the detection of even lower level features such as oriented edges, corners, etc. In this way, CNNs exploit the hierarchical nature of objects in images [28].

After a number of convolutional layers, it is common to make the transition to fully-connected layers (e.g. a normal neural network) which use the detected features to classify or localize the objects in the image.

A 'deep' CNN typically has between 10 and 100 layers, and these layers combined contain sometimes more than one million free parameters. Fortunately, both the weights in the fully-connected layers as well as the filter weights in convolutional layers, can be optimized with a method called gradient descent. In other words, the CNN learns which image features it should extract from the image in order to make the classification in the fully-connected layers work best. This optimization is possible and feasible only, because each weight has a well-defined and analytical partial derivative to the prediction error of the CNN [29].

Apart from these weights, there are however also parameters in the CNN architecture that do not have a well-defined derivative to the prediction error. The reason for this, is mainly that these parameters are discretely valued instead of continuously. Therefore they cannot be optimized with gradient descent. A few examples of these parameters are: the number of convolutional layers, the number of filters in each convolution layer and the size of the convolutional filters. These parameters are called hyperparameters and must be selected manually before the training procedure starts.

Currently filter sizes can only be optimized manually by training multiple CNNs with different filter sizes and seeing what works best. This technique of hyperparameter optimization is in literature often referred to as model (cross-) validation. However this technique is very time consuming in the case of Neural Networks, and finding the optimal set of filter sizes is a combinatorial problem that grows exponentially with the number of filters for which the size needs to be tuned.

1.3. THE IDEA OF THIS RESEARCH

In this report we propose a new filter of which the filter size can be optimized during the training stage. The basic idea for this new filter comes from scale-space theory in which the scale of image features is studied by parameterizing the features with a continuous scale parameter [30]. Thereby decoupling the spatial structure of the image feature from the scale at which it occurs in an image.

$$F[x,y,c] = \sum_{\substack{i \le O,\ j \le O \\ i+j \le O}} \alpha_{i,j,c} \frac{\partial^{i+j}}{\partial x^i \partial y^j} G\left[x,y;\sigma\right] \tag{1.1}$$

The filter proposed in this report, parameterizes a normal filter as a weighted sum of all two-dimensional gaussian derivatives up to a certain order O, see equation 1.1. The resulting effective filter $F[x,y,c]$ is just a normal convolutional filter, but instead of learning these filter weights directly, we learn the weights of the linear combination $\alpha_{i,j,c}$ and the standard deviation of the gaussian derivatives σ that implicitly define the filter weights.

$$k(\sigma) = 2 \lceil 3\sigma \rceil + 1 \tag{1.2}$$

Furthermore, in contrast to normal filters that have a fixed filter size k, the proposed filter has a variable filter size $k(\sigma)$ that depends on the value of sigma, see equation 1.2. The gaussian derivatives are truncated at 3 sigmas from their center (rounded upwards to the nearest integer), because at that point they already contain ~99.7% of their total volume. The advantage of parameterizing normal filters in this way, is that the continuous sigma parameter does have a well-defined derivative to the prediction error, whereas the discrete filter size has not. Hence, gradient descent optimization of the sigma parameter

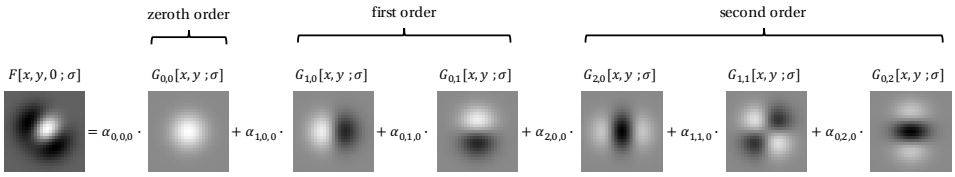


Figure 1.1: An example of how a structured receptive field with one channel is constructed. The resulting effective filter on the left hand side, is a weighted sum of gaussian derivatives on the right hand side. The spatial structure of the filter is completely defined by the weights of the linear combination (the alphas) and the filter size is defined by the standard deviation of the gaussian derivatives (the sigma). The structured receptive field in this example contains all derivatives up to order 2, but this truncation order is arbitrary and can be changed to explicitly control the complexity of the resulting filters.

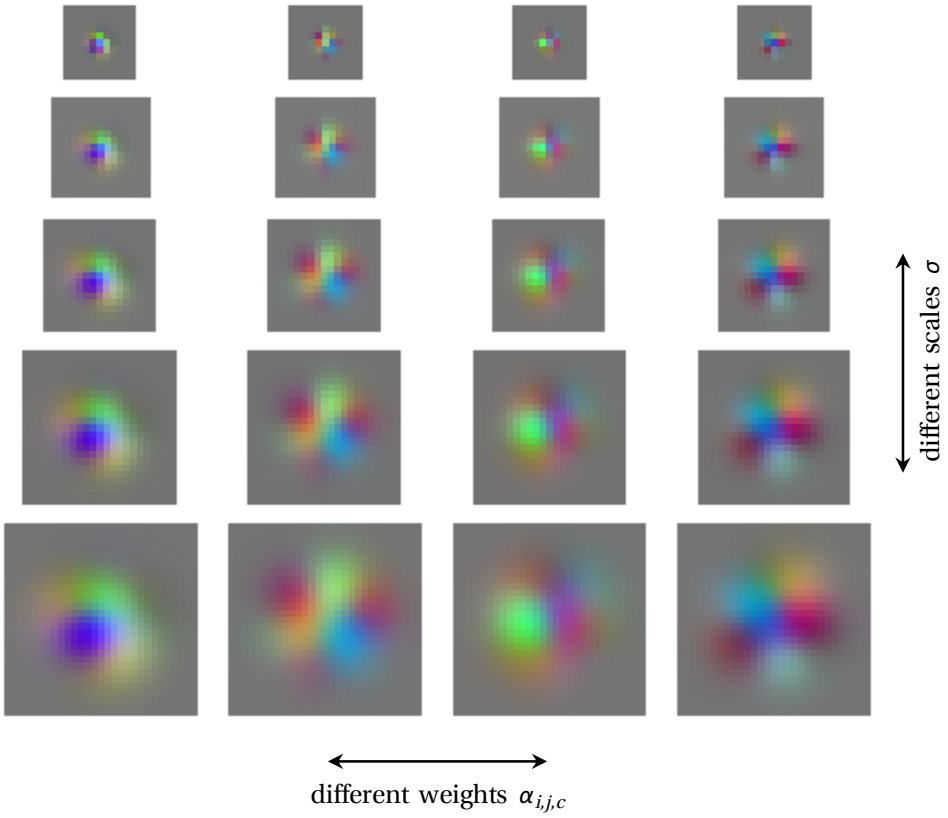


Figure 1.2: This figure illustrates the clear distinction between the role of alpha parameters and the sigma parameter in structured receptive fields. The alpha parameters determine the spatial structure of the filter, while the sigma determines the scale of the filter. Structured receptive fields can, in contrast to normal filters, learn bigger filters without increasing number of learnable parameters and without increasing the complexity of the spatial structure. The complexity of the spatial structure is controlled by the truncation order of the gaussian derivatives.

can be used as a proxy for optimizing the discrete filer size.

Figure 1.1 illustrates how the effective filter is build from a weighted sum of gaussian derivatives. Because the proposed filters are structured by a set gaussian derivatives of which is shown that they can accurately model the response of receptive fields in the visual cortex [30, 31], we will call these filters *Structured Receptive Fields*.

The spatial structure of the filter is determined by the weights of the linear combination and scale is determined by the standard deviation of the gaussian derivatives. This clear and clean distinction between the role of these parameters is illustrated in Figure 1.2. This figure shows that the sigma parameter can change the filter size without changing the spatial structure of the filter, and vice versa. This is a key characteristic of structured receptive fields: in contrast to normal filters, they can learn bigger filters without increasing number of learnable parameters and without increasing the complexity of the spatial structure. The complexity of the spatial structure is controlled by the truncation order of the gaussian derivatives. By including higher order derivatives the filters can represent more complex structures. This is another key characteristic of structured receptive fields: in contrast to normal filters, you can explicitly control the complexity of the spatial structure.

In this research we continue on the work of *Jacobsen et al.* [32] who showed that structured receptive fields are to be preferred over normal filters when only few training examples are available. However in their research, only the structure of the filters (the alpha parameters) was learned and scales were selected manually, while in this research both the structure and the scale will be learned. See the related work in section 1.5, for more a detailed discussion.

1.4. RESEARCH QUESTIONS

The main research question of this report is:

Can structured receptive fields replace normal filters in any CNN architecture with an always strictly better or equal performance in classification tasks?

This main question sets a quite ambitious goal, and we might not be able to answer it conclusively. Therefore we divided the main question into a set of subquestions that we should be able to answer. The goal of these subquestions is to provide a proof of concept that demonstrates the practical potential structured receptive fields. Hence, it should at least bring us a step closer to answering the main question.

Proof of concept

The proof of concept should answer the following subquestions:

1. Can structured receptive fields approximate all the filters that learned by normal CNNs, despite their imposed mathematical form?

2. It is possible to learn the feature scale (filter size) of structured receptive fields during the training stage?
3. Does the usage of structured receptive fields provide an advantage over using normal filters, in classification tasks?

Besides a possible advantage over normal filters in classification tasks, there are extra applications that are unique to structured receptive fields. These applications arise from the fact that structured receptive fields, in contrast to normal filters, contain an explicit and continuous scale parameter that is learned during the training stage. As a bonus we will therefore, also investigate two extra use-cases of structured receptive fields.

Extra applications
The extra applications are investigated by answering the following subquestions:

1. Can the learned scale parameters be adjusted manually after the training, to change the global scale at which the whole CNN operates? I.e. is the CNN able to classify the same image correctly over a range of different scales while being trained on images of a single scale only, if we select the right global scale?
2. Is it possible to use the learned scale to investigate the role of subsampling layers in CNNs? I.e. when subsampling layers are omitted, do the structured receptive fields learn larger feature scales for the subsequent convolutional layers and what happens to classification accuracy?

The subquestions regarding the *Proof of concept* and the *Extra applications* will be answered in chapter 5 and 6 respectively. The main question will be answered in the conclusion in chapter 7. The answers to all the subquestions will also be repeated there, to give a complete overview of all the results in this research.

1.5. RELATED WORK AND HISTORICAL CONTEXT

This section will provide an overview of related work and simultaneously it will try to place this research into historical context.

STRUCTURED RECEPTIVE FIELDS IN PREVIOUS WORK

First and foremost it should be mentioned that this research continues on the work of *Jacobsen et al.* [32], who showed that structured receptive fields have the advantage over normal filters when only few training examples are available. The fact that structured receptive fields have some desirable properties that do not have to be learned, makes them more powerful when the amount training data is limited. While the focus of their research is mainly on the structure of features (in relation to small training sets), this research focusses on the scale of features. More concretely, in their research the structure of the features is learned and scales are selected manually, while in this research both the structure and the scale will be learned.

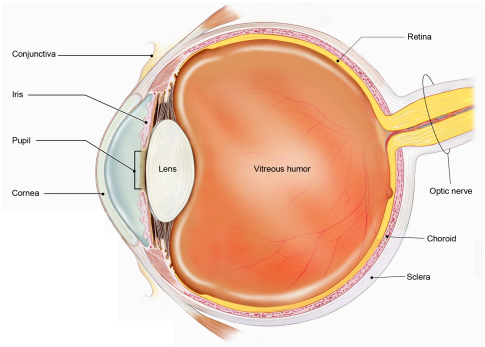


Figure 1.3: Atonomy of the eye, including the retina on which light gets projected (Source: National Eye Institute, NEI)

BIOLOGICAL STUDY OF THE VISUAL SYSTEM

It was shown that gaussian derivatives up to 3rd and 4th order, can accurately model the signals to which cells in the visual cortex respond [30, 31]. Since many of the previous work was inspired by these findings, we will give a brief overview of the workings of the human visual system.

The components involved in the ability of living organisms to perceive and process visual information are referred to collectively as the visual system. The visual system starts with the eye (see figure 1.3), light entering the eye passes through the cornea, pupil and lens. The lens then refracts the light and projects it onto the retina. The retina consists of two types of photoreceptor cells, rods and cones. The rods and cones fulfill a different purpose. The rods are more sensitive to intensity and less to color, while the cones are less sensitive to intensity and more to color [33]. The rods and cones are connected to ganglion cells. Each ganglion cell covers a different region on the retina and therefore receives light from a different spatial location. The regions covered by these ganglions are called *Receptive Fields* [34].

The receptive field of a ganglion cell is organized in two concentric layers, the center and the surround. The ganglion cell fires differently depending on whether light falls on the center, the surround or both. The ganglion cells in the eyes are connected via other cells to the lateral geniculate nucleus (LGN) in the brain. The LGN relays the signals of multiple ganglions to simple and complex cells. By combining the responses of multiple ganglion cells, a more complex response to a bigger receptive field is formed. The workings of this can be seen in figure 1.4. In the 1950's Hubel and Wiesel won a Nobel prize for showing that these complex cells respond primarily to edges and gratings of a particular orientation [35] [2]. The edges and gratings to which complex cells respond can be accurately modeled in terms of Gaussian derivatives up to 3rd-4th order [30, 31].

[2] There is an online video in which Hubel and Wiesel perform their experiment on a cat. It shows how complex cells in the visual cortex respond to edges and gratings of a particular orientation.

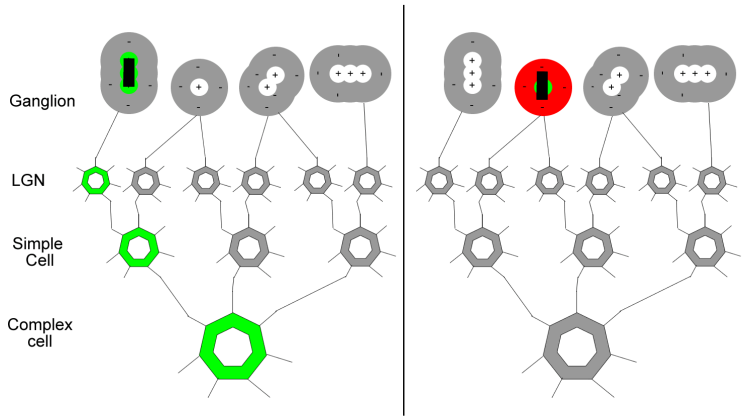


Figure 1.4: On the left we see how the firing of multiple ganglion cells combines to the firing of a complex cell. The firing of three ganglion cells propogates through the LGN via the simple cell to the complex cell. The three ganglion cells form the receptive field of the complex cell, and in this example the complex cell fires as result of the detection of a horizontal edge. On the right it we see that the ganglion cell does not fire because light falls both on the center and the surround.

SCALE-SPACE THEORY

Although never used in combination with CNNs, the idea of describing image features in terms of gaussian derivatives of which the standard deviation serves as a continuous scale parameter is not new. In fact a whole mathematical framework has been developed to study the scale of features in images. This framework is called *Scale-space theory* and was primarily developed in the 1980s and 1990s by *Witkin* [36], *Koenderink* [37], *Lindeberg* [38] , *Florack* [39] and *Romeny*[40]. The framework aims at studying the multi-scale nature of images. This multi-scale nature originates from the fact that the distance from objects to the imaging apperatus (i.e. a CCD cameara, a telescope, a microscope or the human eye) may vary. This varying distance causes the appearance of objects in the image to change inevitably. When the distance becomes too big, small objects may disappear or many small objects may merge into one bigger looking object. Scale-space theory describes the process of imaging of a real physical space at a certain scale in a mathematical way, hence the name *Scale-space theory*.

Starting from a set of desirable properties called scale-space axioms, they showed that only convolution with gaussian kernel exhibits all of the properties needed to describe the imaging process [30, 37]. These properties include amongst others: linearity, shift and rotation invariance and the non-creation and non-enhancement of local extrema as scale increases. The gaussian smoothed images form a scale-space representation, in which the discrete spatial dimensions of the image are extended by a continuous scale dimension. The variance of the gaussian fulfills the role of the scale dimension and is called the scale parameter. In this representation the original image corresponds to the scale dimension being zero. This is called the inner scale of the image and is determined by the imaging apparatus. For example images from a microscope have a different inner

scale, than images made with a telescope. The purpose of the scale-space representation is that the same structures (features) can be detected at different scales if the appropriate scale is selected. It has been shown that scale selection can be done automatically for the structures that are expressed in terms of gaussian derivatives [41]. The procedure basically corresponds to the selecting the scale at which the feature response is strongest. Some examples of these differential structures (features) are: blobs, edges, ridges and corners. Structured receptive fields relate to these features in the sense that they are also expressed in terms of gaussian derivatives.

FILTER SIZES IN NORMAL CNNS

As far as we are aware, this is the first time that the filter sizes are learned during the training stage [3]. In the past however, a lot of research and effort has gone into manually selecting filter sizes. In 2014 the *Visual Geometry Group* (VGG) won the ImageNet competition using a very deep CNN with only 3x3 filters [43]. Their main argument for using 3x3 filters only, is that a stack of multiple 3x3 filters has the same effective receptive field as large filter while containing less parameters. For example, a stack of three 3x3 filters contains 27 parameters and has the same effective receptive field as one 7x7 filter which contains 49 parameters. They argued that using less parameters while still being able to learn large effective filters, has a similar effect as regularizing large filters. This relates to structured receptive fields, which can also increase their filter size without increasing the number of parameters.

A TREND TO USE INCREASINGLY DEEPER CNNS

A difference with the VGG-network is that in the VGG-network the effective filter is distributed of multiple convolution layers with non-linear activation functions in between. They showed that not only the usage of the 3x3 filters, but also the usage of these extra non-linearities makes the network more powerful. In the years to follow, much of the research has shifted from selecting optimal filter sizes towards training deeper CNNs with only 3x3 filters. Much of this research has been devoted to fighting the vanishing gradient problem [4] that comes with these deeper CNNs. Recently some groups were able to reduce this vanishing gradient problem by introducing skip-connections from lower to higher layers. For the sake of completeness, we will mention some of the successful architectures: *Highway* [44], *Residual* [45], *Stochastic Depth* [46] and *Densely Connected Neural Networks* [47]. Densely Connected Neural Networks or simply DenseNets, currently hold the state-of-the-art results on many of the benchmark datasets for classifica-

[3] While working on this thesis, work has been published [42] in which the filter size is also learned during the training stage. However they use a different method in which they linearly interpolate between a virtual border of 1 pixel and the current border of the filter. The key differences with respect to our approach is that their filters are more like normal filters because they are non-parametric, whereas structured receptive fields are parametric. Hence their filters do not decouple the structure from the scale, and they have no way to control the complexity of the structure as the size of the filter increases. Furthermore, they compare their method on different datasets and to different baseline results than this report.

[4] The vanishing gradient problem occurs when derivatives of the prediction error w.r.t. parameters becomes so small that training the parameters with gradient descent takes forever. This typically happens for parameters in the lower layers of very deep convolutional neural networks.

tion, including the SVHN [48], ImageNet [9] CIFAR-10 and CIFAR-100 datasets [49]. In this report, we compare the original DenseNet architecture with a DenseNet architecture in which all normal filters have been replaced with structured receptive fields, on the CIFAR-10 dataset.

GABOR WAVELETS IN CNNS

Last but not least, we should mention that gaussian derivatives are not the only basis filters that can be used to structure filters. For example, in both [50] and [51] *Gabor Wavelets* were used instead of gaussian derivatives. Gabor wavelets look a lot like gaussian derivatives, because they are sinusoids that are modulated by a gaussian envelope. The standard deviation of the gaussian could therefore also serve as a continuous scale parameter. However like in *Jacobsen et al.* [32] they manually selected the scale parameters and the wavelets were only used to structure the filters, and not to learn the feature scale.

1.6. REPORT STRUCTURE

The remainder of the report is used to discuss the findings of this research.

Part I contains a comprehensive introduction to machine learning and artificial neural networks. These chapters are highly recommended for readers that have little or no experience with machine learning. Chapter 2 explains how machine learning uses a data driven approach to build prediction models. Chapter 3 zooms in on a particular group of machine learning algorithms called artificial neural networks. It explains what neural networks are, how they work and why convolutional neural networks are so particularly useful for solving problems involving image data.

Part II describes the experimental-methods, it covers all the implementation details of structured receptive fields. It describes how the filter size of structured receptive fields can be learned, which differences there are compared to normal filters, and how structured receptive fields can be implemented in a convolutional layer, in a computationally efficient manner.

The experiments and results are featured in Part III. Chapter 5 contains three *Proof of concept* experiments. The first experiment investigates the ability of structured receptive fields to approximate any filter learned by normal CNNs. The second experiment questions the extra ability of structured receptive fields over normal filters, to learn their own filter size. The third and final experiment investigates whether structured receptive fields provide an actual advantage over normal filters, when used in classification tasks. Chapter 6 explores two *Extra applications* that are unique to structured receptive fields. The first application explores whether the learned scale parameters can be adjusted after the training stage, to change the global scale at which the CNN operates. The second application explores whether structured receptive fields can be used to investigate the role and the importance of subsampling layers in CNNs.

Part IV wraps up the entire research. Chapter 7 summarizes all the results and conclusions. By doing so, this chapter answers the sub- and main research questions of this report. The importance of these answers for the scientific world are discussed in chapter 8, as well as the limitations of our method. Finally, a few things are mentioned that we would have done differently in hindsight, before we recommend the most promising and interesting directions for future research.

I

THEORY

2

INTRODUCTION TO MACHINE LEARNING

Machine learning is the subfield of computer science that gives computers the ability to learn without being explicitly programmed.

Arthur Samuel, 1959

THIS chapter will explain how machine learning can be used to build a prediction model. A good prediction model can predict a quantity that is not (yet) directly observable/measurable, through a set of input quantities that are directly available. An example is predicting the price of a house based on its square-footage, number of rooms, number of bathrooms, etc.

This chapter starts by giving a general overview of how to build a prediction model. Section 2.2 and 2.4 explain how a machine learning algorithm learns the relationship between the output and input variables, from a dataset of examples. The different branches of machine learning are mentioned in section 2.3, before restricting ourselves to supervised learning branch only. Section 2.5 features the different algorithms that can be used for supervised learning. We will see that Neural Networks constitute a particular group of algorithms. The main issue with learning a relationship from a dataset of examples is called overfitting and will be covered in section 2.7. The final section discusses the recent popularity of Machine Learning and Neural Networks in particular. It will explain which breakthroughs in the development of Neural Networks have led to the current hype of Deep Learning, before explaining what Neural Networks are in the next Chapter.

2.1. BUILDING A PREDICTION MODEL

The goal of building a model is to find the relationship between input and output. This goal is reached once a relationship is found that can make accurate predictions on the output given a specific input. In computer science such a relationship is represented as an algorithm. In mathematics it is described by a function $f(\cdot)$ with input $\vec{x}$ and output $\vec{y}$. In general, $\vec{x}$ and $\vec{y}$ are vectors, since the problem can have multiple inputs and outputs.

$$f(\vec{x}) = \vec{y} \tag{2.1}$$

This simple and very general equation will be the starting point for explaining what machine learning is.

In equation 2.1 the model $f(\vec{x})$ on the left hand side is used to predict the output $\vec{y}$ on the right hand side. The model consists of two parts: the relationship $f(\cdot)$ and the choice of input information $\vec{x}$. Each component of the input vector holds a different piece of information, which is called a feature. For example, if you want to forcast the price of a house you could base your prediction on the following 3 features: square-footage, number of rooms and number of bathrooms. In that case the input would be a 3-dimensional feature vector.

So in order to build a good model, there are two questions that need to be answered:

1. Which features do we need in $\vec{x}$ to be able to make accurate predictions on $\vec{y}$?
2. Given the information $\vec{x}$, what is the relationship $f(\cdot)$ that makes the most accurate predictions on $\vec{y}$?

The answer to the first question depends on the problem that you are trying to solve. For problems with image data however, it is quite common to take each pixel value as a seperate input feature and thereby incorporating every bit of information there is. Since the focus of this research is mainly on image data, I will not elaborate any futher on how to answer this first question. Instead, I will focus on answering the second question: how to find the best relationship $f(\cdot)$ given a certain set of input features $\vec{x}$?

2.2. THE MACHINE LEARNING APPROACH

The idea of machine learning is that the relationship $f(\cdot)$ can be learned purely by observing a large number of example input-output pairs $(\vec{x}, \vec{y})$. In principle it can learn this relationship, without making assumptions on its form nor does it require explicit human instructions or prior knowledge to find it. This defines machine learning as a form of Artificial Intelligence (AI).

2.3. BRANCHES OF MACHINE LEARNING

The examples of input-output pairs from which the machine learns the relationship, is called a dataset. Although a dataset always contains examples of input, it does not

necesserily contain information about the corresponding output. The field of machine learning is subdivided into the following three branches, based on what is known about the output:

- Supervised learning ($\vec{y}$ is known).
- Unsupervised learning ($\vec{y}$ is unknown).
- Reinforcement learning ($\vec{y}$ is currently unknown, but at some moment in the future one will be able to infer what $\vec{y}$ should have been). [1]

The division of these branches is only mentioned for the sake of completeness. In the remainder of this report we will restricted ourselves to datasets where the corresponding output is known. So, this research report will focus on supervised learning only!

2.4. SUPERVISED LEARNING

In supervised learning, it is common to make a distinction between regression problems and classification problems. In regression problems the goal is to predict the outcome of a continuous valued function, whereas in classification problems the goal is to indentify group membership. For example, forecasting the price of a house is a regression problem and identifying a fruit as an apple, an orange or a banana is a classification problem.

Note that regression and classification problems are actually very similar. A classification problem can be formulated as a regression problem of a decision boundary, with the additional step of checking on which side of the decision boundary an object lies. A decision boundary is essentially a continuous valued function (a surface) that seperates two classes. For example, the surface of the earth is the decision boundary that seperates points inside the earth from points outside the earth. Most machine algorithms that can solve regression problems can thus be extended to solve classification problems as well.

But how does one solve a regression problem? How does one find the mathematical function that best describes the relationship between a set of corresponding input-output pairs? In general one starts by picking a family of functions. A family of functions is a function definition that contains free parameters. For each valid set of parameter values it defines a different function. Equation 2.2 shows for example the family of linear functions. The variable x designates the function's argument, but θ_0 and θ_1 are parameters that determine which particular linear function is being considered.

$$f_\theta(x) = \theta_0 + \theta_1 x \tag{2.2}$$

The next step is finding the parameter values that minimize the difference between the model prediction $\hat{\vec{y}}_i = f_\theta(\vec{x}_i)$ and the actual value $\vec{y}_i$ of all examples in the dataset. In literature there are many different names for the process of finding the right parameter

[1] An example of this is gameplay. In games it is sometimes hard to determine what the best action is. However a few steps later in the game, it might be possible to infer whether the action was good or not.

values. Some commonly used names are: optimizing, training, learning, fitting and regressing.

The total amount by which the predictions deviate from the actual values is measured by a loss function, which usually combines errors of all individual examples into a single number. [2] There are many different loss functions, but the most common are the cross entropy loss for classification problems and the mean squared error (MSE) for regression problems. [3]

So, learning a relationship simply boils down to picking the right family of functions and finding the best values for its free parameters. In supervised learning, these parameter values can be found by minimizing a loss function over a dataset with labeled examples. But how does one choose the right family of functions?

2.5. DIFFERENT SUPERVISED LEARNING ALGORITHMS

This is a good moment to introduce the different supervised machine learning algorithms. I will start by presenting an extensive but by no means complete list of the different categories and subcategories of supervised machine learning algorithms:

- Regression algorithms.
  - Linear regression. [4]
  - Logistic regression.
- Bayesian classifiers.
  - Naive bayes classifier.
  - Nearest mean classifier.
  - Linear discrimant analysis.
  - Quadratic discrimant analysis.
  - Parzens classifier.
- K-nearest neighbours algorithms.
- Decision trees.
  - Random forests
- Support vector machines (SVM).
  - Linear SVM.
  - Non-linear SVM.
- Artificial Neural networks (ANN).
  - Recurrent networks.
  - Feedforward networks.
    - Fully-connected networks.
    - Convolutional networks.

The list might seem quite overwhelming, but it is important to realise that each of these algorithms is fundamentally nothing more than a particular family of functions with free parameters and that these parameters can be optimized to change the behaviour of the algorithm. Figure 2.1 serves as an illustration of this idea.

[2] Almost all loss functions can be writen as a sum over the losses of the individual training examples. A loss function of this form is required when using batch gradient descent to optimize parameters.
[3] Some other loss functions are the zero-one loss, the hinge loss and the Kullback–Leibler divergence.
[4] Linear regression is probably the simplest and most known supervised machine learning algorithm. However people usually don't realise that it is a machine learning algorithm.

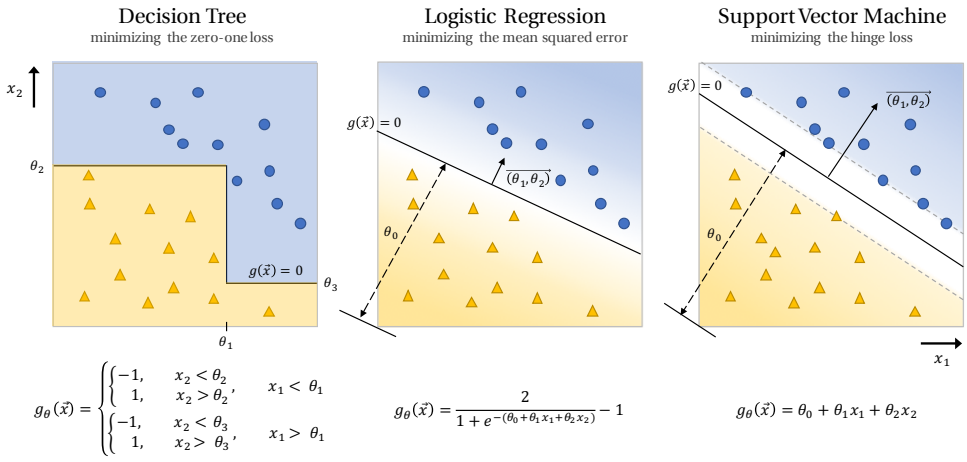


Figure 2.1: The same two-dimensional (x_1, x_2) and binary (triangle vs. circle) classification problem solved by three different machine learning algorithms: a decision tree, logistic regression and the support vector machine. As usual in classification problems, first a decision function $g_\theta(\vec{x})$ is regressed and the final classification is done by checking on which side of the decision boundary an object lies. The decision boundary is the line $g(\vec{x}) = 0$ and the color coding of the different regions reflects how certain the classifier is that an object lying in that region, belongs to a particular class. Geometric meaning of the free parameters is shown for the sake of completeness. Because of the different decision functions and the fact that they are all combined with different loss functions, each decision boundary is an optimum of a slightly different objective.

In Figure 2.1 the same two-dimensional (x_1, x_2) and binary (triangle vs. circle) classification problem is solved with three different machine learning algorithms. As usual in classification problems, first a decision function $g_\theta(\vec{x})$ is regressed and the final classification $f_\theta(\vec{x})$ is done by checking on which side of the decision boundary an object lies. Although each algorithm has a different decision function, they all contain free parameters. So a supervised machine learning algorithm is nothing more than a family of functions with free parameters that can be optimized by minimizing a loss function over a dataset with labeled examples.

The optimization techniques that can be used to find the optimal parameter values, are determined by the loss function and the particular algorithm that is being used. They will also determine whether finding a global minimum can be garuanteed or not. For example, in the case of linear regression with a mean squared error finding a global minimum can be garantueed. The parameter values can be computed via a closed form solution which is known as the normal equations. In the case of the linear SVM with a hinge loss, a global minimum can also be garantueed. The Karush–Kuhn–Tucker conditions transform the minimization problem with constraints into a constraint free minimization problem that can be solved with a technique called quadratic programming. So the training of a machine learning algorithm is just an optimization problem (possibly with contraints) and all the theory from that subfield of mathematics applies. In the next chapter we will see that for Neural Networks a global minimum cannot be garantueed,

and that batch gradient descent is the only feasible technique that can be used to optimize parameter values.

2.6. CHOOSING THE RIGHT ALGORITHM

The choice of algorithm ultimately determines which shapes the decision boundary can take on. In figure 2.1, it is seen that the decision boundary of a decision tree takes on a different shape than the ones of logistic regression and the SVM. Therefore, it is difficult to say which algorithm one should use, because it depends on how the examples of the classes are distributed in the feature space and these distributions are usually hard to visualize because the data is very high dimensional. Algorithms with many free parameters, can form more complex shaped decision boundaries and this enables them to seperate classes that are clustered in more complex ways. This makes them more powerful than simple algorithms because they can be applied to a wider variety of problems. However, when an overly complex algorithm is used a problem will arise that is called overfitting. Overfitting is a problem that is inherent for the way in which a relationship is learned in machine learning. It results from the fact that the algorithm can adapt itself too much to the pecularities of examples in the training set, while you actually want it to generalize well to new and unseen examples. The problem of overfitting will be explained in more detail in the next section.

2.7. MODEL PERFORMANCE AND UNDER- AND OVERFITTING

In supervised learning, the labeled dataset that is used to train the model is called a training set. Applying the trained model to this dataset is obviously of little interest, since the answers of these examples are already known. Instead the main goal is to find a model that generalizes well to unseen data. That means that it make accurate predictions on unseen samples of which the answers are unknown. It is therefore very common to measure the performance of the trained model on a seperate dataset called a test set. Just like the training set, test set examples are labeled with the right answer the only difference is that these answers are not used for training the model parameters, but only for measuring the model performance.

The main objective of maximizing the performance on the test set does not always correspond well to minimizing the loss function over the training set. For complex algorithms that have many free parameters, the performance on the training set is usually a lot better than on the test set. In order to understand this, we will make the distinction between learning general rules and learning specific rules. General rules apply to almost all examples in the training set and specific rules apply only to certain examples in the dataset. These specific rules are usually unwanted, because they only cover the pecularities of the training set and therefore make generalization to unseen samples harder.

Figure 2.2 shows the decision boundary of three algorithms that were trained using the same dataset. Each algorithm has a different level of complexity: a different amount of free parameters. The decision function on the left is so simple that it cannot even learn

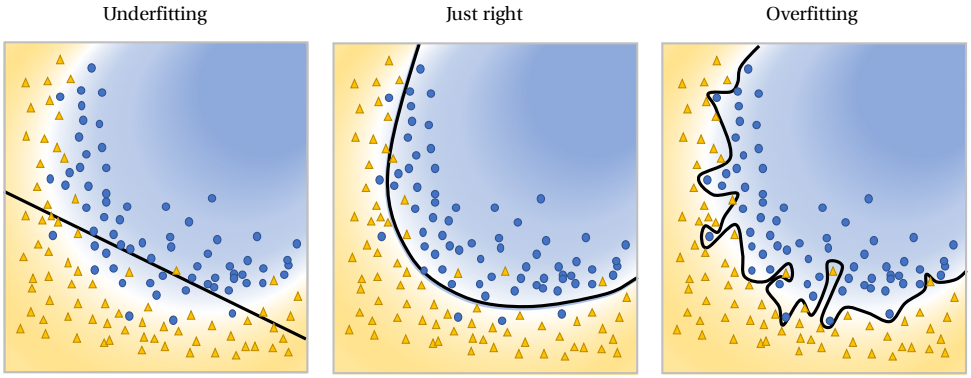


Figure 2.2: An example of three decision boundaries with different levels of complexity, that are trained on the same dataset. The decision function on the left is so simple that it cannot even learn general rules, while the decision function on the right is so complex that it can learn specific rules on top of the general rules. The decision function in the middle is just right and probably has the best performance on new samples that are not in the training set.

general rules, while the decision function on the right is so complex that it can learn specific rules on top of the general rules. The decision function in the middle is just right and probably has the best performance on new and unseen samples. So when a model is too simple it will perform badly on both the training and the test set, and when it is too complex it will perform very good on the training set but suboptimal on the test set. These two scenarios are respectively called underfitting and overfitting.

It is important to realize once more, that it is usually hard to visualize the data because it is very high dimensional. So figuring out whether the algorithm is underfitting or overfitting really comes down to measuring and comparing the performance on the training and test set.

2.8. THE RECENT POPULARITY OF MACHINE LEARNING

The fact that machine learning does not require human prior knowledge to find a relationship, makes it very powerful in situations where this knowledge is absent [5]. For example, when you suspect a relation between input and output variables but you have no idea how to model this relationship. The procedure for finding this relationship with machine learning is quite straightforward:

1. Collect a dataset with input-output examples.
2. Pick a supervised machine learning algorithm and define a loss function.
3. Let the optimizer find the parameter values that minimize this loss function.

[5] Although prior knowledge is not required for finding the relationship, it is still very important for choosing the right features!

Note that this procedure does not contain any problem specific assumptions and is therefore applicable to a wide variety of problems.

We currently live in world of *Big-Data*, which means that collecting a large dataset is usually not very hard. These large datasets enable the training of more complex algorithms without overfitting them on the training set. These complex algorithms are capable of solving more complex problems that had not been solved before. Some examples of these problems are: beating humans at the game of Go, building cars that can drive autonomously and automatic diagnosis of patients based on medical images. Recent developments in *Distributed computing*, *Cloud computing* and *Parrallel programming* on the GPU, have provided all the tools needed for training these complex algorithms in a time efficient manner. All of the above has resulted in a recent growth in popularity of machine learning algorithms and neural networks (*Deep Learning*) in particular.

The reason why neural networks are so powerful lies in the fact that they can approximate any continuous function [20]. For a classification problem this means concretely that the decision boundary can take on any shape, while the shapes of other algorithms are restricted to a particular set. This arguably makes neural networks the most powerful algorithm of all machine learning algorithms: it can learn any relationship without making assumptions on the mathematical form. [6] However this does mean that they suffer a lot from the problem of overfitting. In the next chapter we will see what neural networks are and how the problem of overfitting is substantially reduced for problems involving image data, when convolutional neural networks are used.

[6] Keep in mind that selecting the input features is also a very crucial part of building a prediction model! For example, it is impossible to give a very precise estimate of the price of a house, when that estimate is based solely on the color of the front door.

3

INTRODUCTION TO CONVOLUTIONAL NEURAL NETWORKS

I think people need to understand that deep learning is making a lot of things, behind-the-scenes, much better.

Geoffrey Hinton

THIS chapter will give an introduction to a particular group of machine learning algorithms called: Artificial Neural Networks. There are several types of neural networks, but the main focus of this chapter will be on Convolutional Neural Networks. In this chapter we will see what neural networks are, how they work and why convolutional neural networks are so particularly useful for solving problems involving image data.

The chapter starts by describing how the working of the human brain, served as an inspiration to come up with artificial neural networks as a single algorithm that can be tuned for a wide variety of different tasks. The neuron is the main building block of neural networks, and is covered in section 3.2 and 3.3. A neuron is a simple processing unit that takes incoming signals from other neurons, processes them, and sends an outgoing signal to other neurons. In section 3.4 we will see that different types of neural networks can be formed by connecting neurons in different patterns. Section 3.5 features fully-connected neural networks as a prelude to explaining convolutional neural networks in section 3.6. We will see that fully-connected neural networks do not work well when applied directly to images, and that the local connectivity pattern of convolutional neural networks solves these problems by exploiting the spatial structure of images. The final sections describe the training of neural networks, before continuing to the next chapter.

3.1. A LEARNING ALGORITHM INSPIRED BY THE HUMAN BRAIN

In the previous chapter, it was explained that supervised machine learning algorithms are mathematical functions that contain free parameters, and that the right values for these parameters can be found by minimizing a loss function over a dataset of labeled examples.

There are many different types of supervised machine learning algorithms. The precise mathematical structure of the algorithm determines which types of relationships it can learn. I find it convienent to imagine the type of relationship as the shape of a decision boundary in a classification problem. A linear function can only produce a decision boundaries that are straight lines. These straight lines can be rotated and translated by changing the parameters, but it can never be transformed into an ellipse. A quadratic function on the other hand, can produce ellipses but also a straight lines by setting the parameters of the quadratic terms to zero. This leads to the question: is there an algorithm that can learn every possible decision boundary? In other words, is there an algorithm that can approximate every possible mathematical relationship? The answer is yes and it is the Artificial Neural Network (ANN).

The artificial neural network is an algorithm that is biologically inspired by the human brain. Neuroscientists discovered that after a stroke, other brain parts can take over the functions of the lost part. However this does not happen automatically. Instead the patient needs to re-learn these functions. This caused AI researchers to believe that the different brain parts can be modelled by exactly the same algorithm and that this algorithm can be tuned for a particular task by changing the values of its parameters. They called this algorithm the Artificial Neural Network.

3.2. A NETWORK OF NEURONS

The artificial neural network is a mathematical function that aims to describe how the human brain processes information. The brain is a biological neural network, that consists of a series of interconnected neurons. Each neuron is a simple processing unit that takes incoming signals from other neurons, processes them, and sends an outgoing signal to other neurons. Although each individual neuron is quite simple, by connecting trillions of these simple neurons a complex network is formed that we call the human brain.

The next section will explain the workings of a biological neuron in more detail. It will also explain that an artificial neuron is a mathematical description of how the biological neuron works.

3.3. BIOLOGICAL AND ARTIFICIAL NEURONS

Figure 3.1 shows a schematic view of a biological and an artificial neuron. The biological neuron receives incoming signals from other neurons via its *Dendrites*. These signals

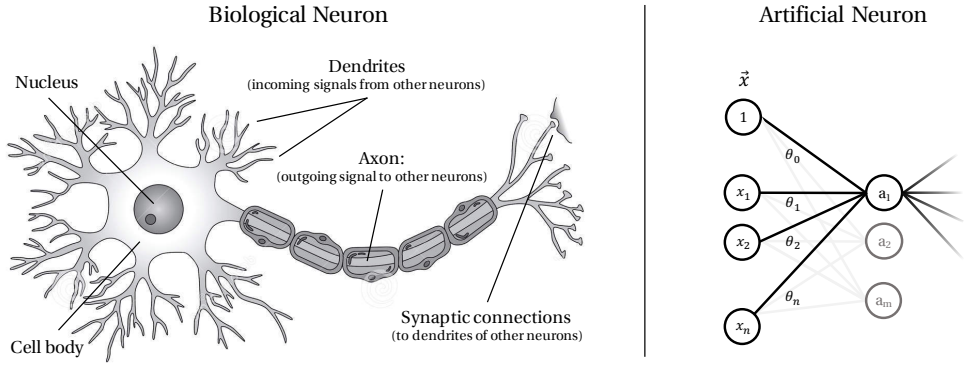


Figure 3.1: A schematic of a biological neuron on the left, and a schematic of an artificial neuron on the right. The biological neuron receives incoming signals from other neurons via its Dendrites. These signals are accumulated in the Cell Body. When this accumulated signal surpasses a certain threshold, the Axon fires an output signal. This outgoing signal flows via Synaptic connections to the Dendrites of other neurons where the proces repeats itself. The artificial neuron is a mathematical description of how the biological neuron works. In this schematic each node represents a neuron and the connections between them represent the synaptic connections.

are accumulated in the *Cell Body*. When this accumulated signal surpasses a certain threshold, the *Axon* fires an output signal. This outgoing signal flows via the *Synaptic connections* to the *Dendrites* of other neurons, where the proces repeats itself.

The strength of the synaptic connection between the Axon and the Dendrite of the other neuron is called a synaptic weight. It reflects how much influence the firing of the sending neuron has on the receiving neuron. The free parameters in the artificial neural network are supposed to represent these synaptic weights. But how these synaptic weights are changed inside the brain and how humans learn is only poorly understood.

An artificial neuron is a mathematical description of how the biological neuron works. Its connectivity to other artificial neurons is often depicted in schematics such as the one in figure 3.1. In this connectivity scheme each node represents a neuron and the connections between them represent the synaptic connections. For this example, we focus on the neuron a_1. The synaptic weights corresponding to the input signals x_i are denoted by θ_i and the output signal is denoted by a_1. The working of this artificial neuron is described by the function in equation 3.1.

$$a_1 = \phi\left(\theta_0 + \sum_{i=1}^{n} \theta_i x_i\right) \tag{3.1}$$

The artificial neuron takes a weighted sum of the incoming signals (including an offset term θ_0) and applies an activation function $\phi(\cdot)$ to it. This activation function serves the purpose of mimicking the firing behaviour of a biological neuron. Many different kinds of activation functions have been used in the past, but nowadays the most commonly

used activation function is the Rectified Linear Unit (ReLU), see equation 3.2. This activation function acts as a gate by letting positive inputs go through unmodified and setting negative inputs to zero. The output signal of the activation function is then send to other connected neurons, where the proces repeats itself.

$$\phi(x) = \max(0, x) \tag{3.2}$$

An artificial neuron is of course only a rough description of how a biological neuron in the brain works. For example, the relu activation function does not describe the firing behaviour correctly. But giving a complete and correct mathematical description of the human brain, was never the intention of the AI researchers. That job is reserved for the field of Neuroscience. Instead studying the workings of the human brain, served as an inspiration to come up with a single algorithm that can be tuned for a wide variety of different tasks. From this moment on forward, we will take a purely mathematical viewpoint on artificial neural networks and we will call them simply, neural networks.

In the next section we will see that we can form different types of neural networks by connecting the neurons in different schemes.

3.4. DIFFERENT TYPES OF NEURAL NETWORKS

A neural network can be represented schematically as a directed graph in which the nodes are the neurons and the edges are the connection between these neurons. See figure 3.2 for an example. The direction and weights of the edges, represent the direction in which information flows and how important that information is to the receiving neuron. There are some common patterns in which neurons are usually connected. These different patterns constitute the different types of neural networks.

It is common to make a first distinction between recurrent and feedforward connectivity. Recurrent connectivity means that the network contains cycles and feedforward connectivity means that the network does not contain cycles. Recurrent neural networks are used to analyze sequential data in a step-by-step manner. The cycles in the network, give the network the ability to store information about the previous steps. Unlike feedforward neural networks, recurrent neural networks can use this internal memory to process arbitrary sequences of inputs. For example in natural language processing, recurrent neural networks are used to analyze full texts in a word-by-word manner. The interpretation of the current word being analyzed, depends on all the previous words in the text. However, recurrent neural networks are only mentioned for the sake of completeness. This research is about convolutional neural networks, which is a type of feedforward network.

In the next section we will first explain fully-connected neural networks, before moving on to convolutional neural networks.

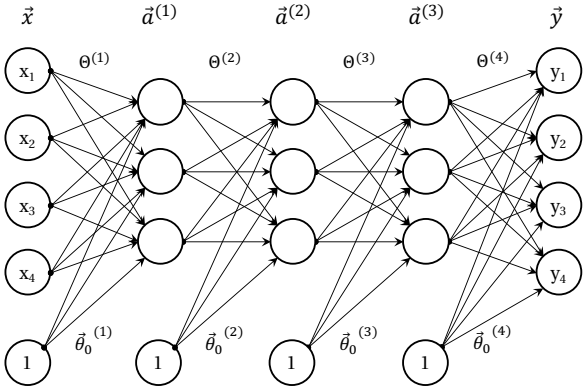


Figure 3.2: A fully-connected neural network schematically visualized as a directed graph. The nodes in the graph are the neurons and the edges represent the connections between these neurons. Except for the input nodes, each node applies an activation function to the sum of the weighted incoming signals. The output of this activation function is send forward to other neurons. This proces repeats itself in a feedforward manner (layer-by-layer) through the network from left to right.

3.5. FULLY-CONNECTED NEURAL NETWORKS

A fully-connected neural network is a type of feedforward network that contains neurons that are organised in hierachical layers. Figure 3.2 shows an example of a 4-layer fully-connected neural network. We say that the neurons are fully-connected, because every neuron in a layer connects with a certain weight to every neuron in the previous layer. Hence the name, fully-connected neural network.

Equation 3.1 in section 3.3 decribes how to compute the output of a single neuron which is a scalar value. However, in fully-connected neural networks it is more convenient to represent the outputs of all neurons in the same layer as a vector. The vector representation allows us to use linear algebra to express the output of the current layer $\vec{a}^{(l)}$, in terms of the output of the previous layer $\vec{a}^{(l-1)}$, see equation 3.3.

$$\vec{a}^{(l)} = L_\theta\left(\vec{a}^{(l-1)}\right) = \phi\left(\vec{\theta}_0^{(l)} + \Theta^{(l)}\vec{a}^{(l-1)}\right) \tag{3.3}$$

Let M be the number of neurons in the current layer and N the number of neurons in the previous layer. Then $\Theta^{(l)}$ is a $M \times N$ matrix in which θ_{ij} is the weight that connects the i^{th} neuron in the previous layer to the j^{th} neuron in the current layer. This matrix-vector product produces a vector which is offsetted by a bias vector $\vec{\theta}_0^{(l)}$, before the activation function $\phi(\cdot)$ is applied elementwise to each component of the resulting vector.

We now have a mathematical description of a high-level building block, which is called a fully-connected layer. The total network can be written as a composite function of these fully-connected layers, see equation 3.4.

$$\begin{aligned}\vec{y} = f_\theta(\vec{x}) &= \left(L^{(l_{max})} \circ \ldots \circ L^{(l)}_{\theta^{(l)}} \circ \ldots \circ L^{(1)}_{\theta^{(1)}} \circ L^{(0)}\right)(\vec{x}) \\ \text{where} \quad L^{(0)} &= \vec{x} \quad \text{and} \quad L^{(l_{max})} = f_\theta(\vec{x})\end{aligned} \tag{3.4}$$

$\circ$ stands for composition: $(f \circ g)(x) = f(g(x))$

For the purpose of illustration, we will also write out the full function for the 4-layer fully-connected neural network in figure 3.2. See equation 3.5.

$$\vec{y} = f_\theta(\vec{x}) = \phi\left(\vec{\theta}^{(4)}_0 + \Theta^{(4)}\phi\left(\vec{\theta}^{(3)}_0 + \Theta^{(3)}\phi\left(\vec{\theta}^{(2)}_0 + \Theta^{(2)}\phi\left(\vec{\theta}^{(1)}_0 + \Theta^{(1)}\vec{x}\right)\right)\right)\right) \tag{3.5}$$

This equation clearly demonstrates the feedforward nature of fully-connected neural networks. The output of the previous layer is fed-forward as input for the next layer.

The layers between the input and output layers are called hidden layers. The network in figure 3.2, consists of 3 hidden layers each containing 3 hidden neurons. However this is just an example. A fully-connected network can have an arbitrary number of hidden layers and each hidden layer can have an arbitrary number of hidden neurons. How many layers and how many neurons one needs, depends on the difficulty of specific problem that one tries to solve. More layers and more neurons per layer, obviously means that the network can learn more complicated relationships. However, bigger networks are also more prone to overfitting and therefore require more training data or other techniques/tricks that reduce the overfitting , see section 3.8.

There is a theorem that has an existential prove, that every possible relationship can be represented by a fully-connected neural network that has a sufficient number of neurons and layers. This theorem is called the *Universal Approximation theorem* and is explained in more detail in the intermezzo below.

The Universal Approximation Theorem

In 1991 Hornik proved the universal approximation theorem [20]. This theorem states that a fully-connected neural network with a single hidden layer containing a finite number of neurons, can approximate any continuous function under mild assumptions on the activation function. Note that this theorem can be transferred to multi layer networks, since they are composed of multiple single hidden layer networks. The universal approximation theorem proves that fully-connected neural networks, can learn every possible mathematical relationship that is continuous. We say that that fully-connected neural networks are universal function approximators [a].

There are two remarks to be made about the universal approximation theorem:

- The theorem proves the existence of an approximation, but it does not say how to find the associated parameter values.
- The theorem mentions convergence can be reached with a finite number of neurons, but it does not say anything about the rate of convergence as function of the number of neurons.

The first remark highlights why the theorem is often misused: it can trick people into thinking that neural networks solve it all. However finding the right parameter values that makes the neural network generalize well to new and unseen examples, is hard!

The second remark touches upon the ongoing discussion of shallow and wide versus deep and narrow networks. Deep neural networks have many layers, while wide networks have many neurons per layer. It is a long conjectured idea that deep (and narrow) neural networks are more efficient at learning complex relationships than shallow (and wide) neural networks. This is the reason why often deep neural networks are used [b], and why it is called *Deep Learning*.

[a] Other universal function approximators are the Taylor series and the Fourier series. However one needs detailed information about the original function to find those approximations, while neural networks can be trained by only using training data.

[b] In the case of fully-connected neural networks, a three layer usually outperforms a two layer network but using more than three layers rarely helps. This is in stark contrast to convolutional neural networks, where network depth has been found to be an extremely important. One argument for this observation, is that images contain lots of hierarchical structure. For example: faces are made up of eyes, which are made up of edges, etc.

Fully-connected neural networks do not work well when applied directly to images, especially when the images are large. Due to full-connectivity pattern they suffer from the curse of dimensionality: meaning that the number of required computations does not scale well with the number of input neurons. A reasonable size for a color image is 1,000x1,000x3 (width, height, color channels respectively). Using the pixels of this image directly as input, means that a single fully-connected neuron would have 3,000,000 weights. This is already a huge number and we are only talking about a single hidden neuron! For a reasonably sized fully-connected neural network, it will simply be impossible to compute the output.

Apart from the computational infeasibility, it is clear that in the case of images, the full-connectivity pattern is a waste of memory and computation time. Because it is not very likely that pixels that are far apart have any statistical relationship. The fact that these pixels are nevertheless connected, means that fully connected neural networks will use these weights to overfit on the examples in the training set. So, even when fully-

connected neural network are used on very small images, they will suffer a lot from overfitting.

In the next section we will see how a convolutional layer solves these problems, by using a local-connectivity pattern that exploits the spatial structure of the images.

3.6. CONVOLUTIONAL NEURAL NETWORKS

A Convolutional Neural Network (CNNs) is a type of feedforward network that takes advantage of the spatial structure of its input signal, often an image. Images are spatially structured signals, meaning that not only the particular values but also the spatial arrangement of pixels contains information. For example, if all pixels of an image were randomly shuffled one could no longer tell what is in the image. Given the importance of this spatial arangement, it makes sense to preserve it and to exploit its properties when building a neural network architecture.

CONVOLUTIONAL LAYERS

The main building block of a CNN is the *Convolutional layer*. Like it is convenient to think of neurons in fully-connected layers as vectors, it is convenient to represent the neurons in convolutional layers as 3-dimensional tensors with spatial dimensions: width × height × channels. For a color image in the input layer, these channels could be the RGB color channels.

A convolutional layer transforms a 3D input tensor into a 3D output tensor. To avoid confusion, we refer to the output channels as featuremaps and to the input channels simply as channels. The name 'featuremap' is particularly well chosen: the output values of neurons in a featuremap indicate the absence or presence of a particular image feature at a particular spatial location in the image. Neurons in different featuremaps detect different image features, while neurons in the same featuremap detect the same image feature, but at different spatial locations. One could imagine an image feature as an small image patch of which the content displays an object of interest. For example, a nose when you want to detect faces.

Now that we have established a high-level description of how a convolutional layer works, lets look at some low-level details. We will first focus on a single featuremap, see the left part of figure 3.3. Each output neuron in the featuremap is connected to a local neighborhood of input neurons in the input tensor. The weights of these local connections are the same for all output neurons, and the neighborhoods of neighboring output neurons partially overlap. So instead of thinking of these local connections as being static, it more convenient to think of them as a filter which slides through the input tensor. At each location the filter takes a weighted sum of all neurons in its neighborhood (also a featuremap specific offset term is added and the activation function is applied). By evaluating this filter at every possible location in the input tensor, a 2-dimensional featuremap is constructed in which spatial order is preserved. The left part of figure 3.3

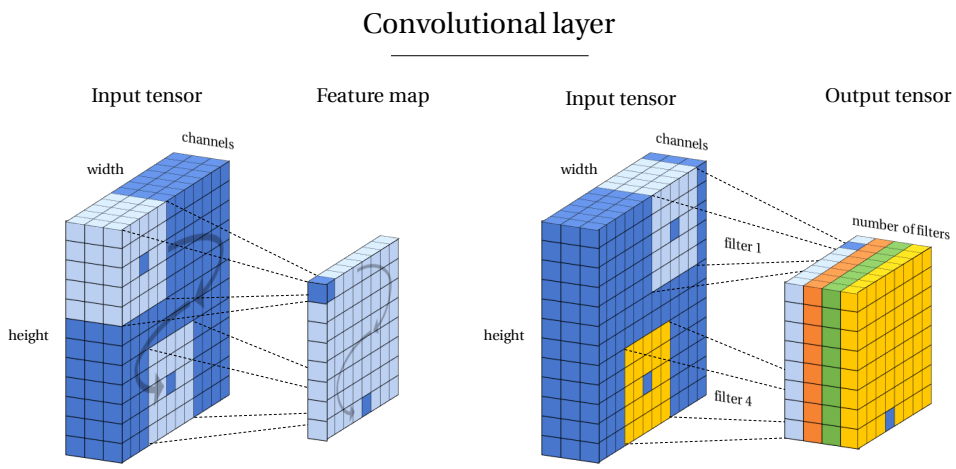


Figure 3.3: Example of how a convolution layer works. The left part shows how a 5x5x3 filter slides through the input tensor and produces an output featuremap. Each output neuron in the featuremap takes as input the signals coming from a local neighborhood of 5x5x3=75 input neurons. A convolutional layer outputs a stack of multiple featuremaps which are all constructed with different filter weights. This can be seen in the right part of the figure.

shows how a 5x5x3 filter slides through the input tensor and produces an output featuremap, in which each neuron takes as input the signals coming from 5x5x3=75 input neurons. The mathematical operation in which a sliding filter computes weighted sums of local neighborhoods, is called *Convolution* [1]. Hence the name, convolutional layer.

There are quite a few details regarding the convolutional layer. We will briefly mention them here. A convolutional layer outputs a stack of featuremaps which are all constructed with different filter weights, see the right part of figure 3.3. A filter only slides along the spatial dimensions (width and height) of the input tensor, and always extends along the full length of the input channels. This ensures that all features detected by the previous layer, can be combined to detect new features in the current layer. For example, if the previous layer contains featuremaps that have detected eyes, ears, mouths and noses, then the current convolutional layer can combine these features to detect faces.

The stepsize with which the filter slides through the input tensor, is called the stride. The stride in the horizontal and vertical direction is usually the same and almost always equal to one. All filters in the same convolutional layer have the same size. The filter size is usually the same in both directions and is usually an odd number to ensure that each filter has a well-defined center pixel. Boundary collisions between the filter and the input tensor, cause the spatial dimensions of the output tensor to shrink. It is common

[1] Strictly speaking it only called convolution when the filter is rotated by 180-degrees. If the filter is not rotated it is actually called cross-correlation. For some reason, the whole community uses cross-correlation while reffering to it as convolution.

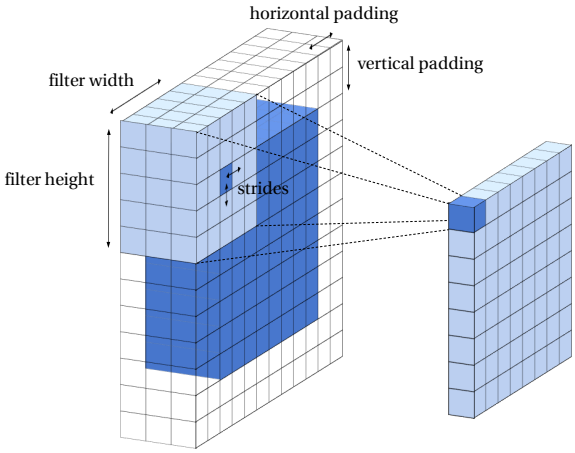


Figure 3.4: The various settings of a convolution layer: the filter size, the strides and the padding. It is most common to use: strides of one, odd filter-sizes and a padding strategy that keeps the dimensions output tensor equal to that of the input tensor. Usually, these settings have the same values in horizontal and vertical direction.

to solve this problem by padding the input tensor with a boundary of zeros before doing the filtering. See figure 3.4, for an illustration of the various settings.

In this report we will use a stride of one, unless explicitly stated otherwise. Furthermore, we will use a padding strategy that keeps the dimensions output tensor equal to that of the input tensor. For these particular settings, equation 3.6 describes how to compute the value for a single output neuron at location x, y in the n^{th} feature map.

$$a^{(l)}[x,y,n] = \phi\left(\theta_0^n + \sum_{f_x=0}^{F_w-1}\sum_{f_y=0}^{F_h-1}\sum_{c=0}^{C-1} a^{(l-1)}[x+f_x, y+f_y, c] \cdot \Theta^n[f_x, f_y, c]\right) \quad \text{for} \quad 0 \le x < W, \quad 0 \le y < H, \quad 0 \le n < N \tag{3.6}$$

In this equation $a^{(l-1)}$ and $a^{(l)}$ represent the 3-dimensional input and output tensor, respectively. The spatial dimensions of both tensors is $W \times H$. The input tensor contains C channels, and the output tensor contains N feature maps. So there are N filters with a filter size of $F_w \times F_h \times C$, each indicated by Θ^n. The bias offset of the n^{th} feature map is denoted by θ_0^n and $\phi(\cdot)$ stands for the activation function.

Summarizing all the differences with fully-connected layers. The convolutional layer preserves spatial structure by aranging the neurons in 3-dimensional tensors. It exploits the strong statistical dependency between nearby pixels, by connecting output neurons only to a local neighborhood of input neurons. It also exploits the translational invariant nature of images (shifting the pixels does not change the information content), by using the same filter weights for every location in the same featuremap. The local-connectivity

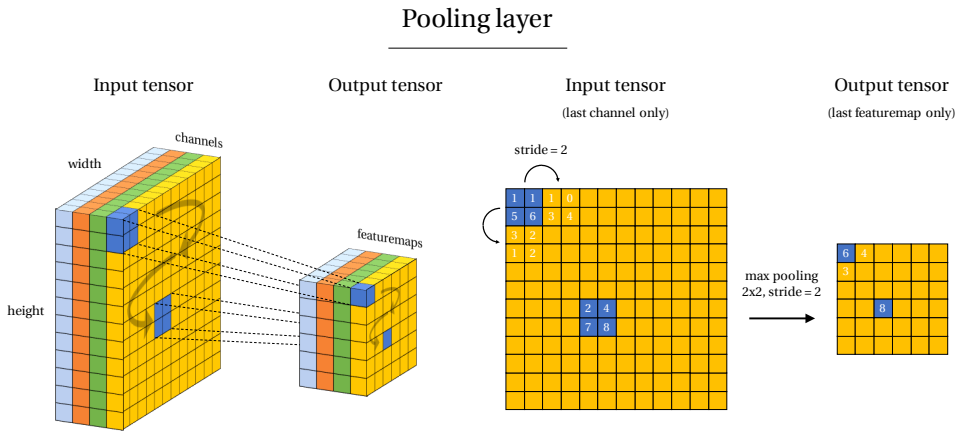


Figure 3.5: Example of how a pooling layer works. This example shows the max pooling layer in it most common form: a 2x2 neighborhood with a stride of two downsamples every input channel by a factor two along both width and height. It takes a maximum over a local neighborhood of 2x2 and strides this neighborhood with a stepsize of two. The left part shows the overall working of the pooling layer, while the right part displays the working on an individual featuremap.

pattern greatly reduces the number of connections, and the weight sharing reduces the number of parameters even further. These adjustments to fully-connected layers lower the computational and memory requirements substantially, while improving the generalization performance of convolutional layers.

This section explained the most important building block: the convolutional layer. In the next section we take a look at the pooling layer, which is another commonly used building block.

POOLING LAYERS

A pooling layer is a commonly used building block in CNNs. It reduces the spatial dimensions (the width and the height) of a tensor by a applying a subsampling operation to each input channel, indepently. There are many variants of this subsampling operation but the most common variants are average and max pooling. The pooling layer works in a similar way as the convolutional layer: it slides a local neighbordhood over the input channel, but instead of taking a weighted sum of this local neighboorhood it takes the maximum or the average. To produce a smaller output tensor these neighborhoods are strided with a stepsize that is equal or greater than two. Summarizing the differences with a convolutional layer: the local neighborhood covers a single channel only and does not extend along the full length of the input channels; the local neighborhoods are strided with a stepsize that is equal or greater than two; the pooling layer has no learnable parameters and computes a statistic (the average or the maximum) instead of a weighted sum.

Figure 3.5 shows an illustration of how a pooling layer works. This example shows a max pooling layer in it most common form: a 2x2 neighborhood with a stride of two downsamples every input channel by a factor two along both width and height. In the downsampled featuremap the maximum value of every 2x2 neighborhood in the input tensor is preserved, while the other three activations are discarded. There are only two commonly seen variations of the pooling layer in practice: a pooling layer with neighborhood of 3x3 and a stride of two (also called overlapping pooling), and more commonly a neighborhood of 2x2 and a stride of two. Pooling with larger sizes and strides are too destructive. It is worth noting that there is currently also a trend to use normal convolutional layers with a stride of two instead of pooling layers [52].

Although the subsampling works well in practice, it is probably one of the least understood parts of CNNs. There are several motivations for using pooling layers in CNNs, but strong empirical evidence to support these motivations has not yet been provided. The original motivation that was given for using max pooling, is that it makes the detection of image features invariant to small local translations [53]. Another important motivation is that the subsampling reduces the dimensionality of the feature space, such that fully-connected layers can be used in the final layers to do the actual classification. A certain advantage of the subsampling is that it reduces the computational and memory load of the subsequent layers, and therefore allows the training of deeper neural networks.

Later in this report, we will show that subsampling enables the detection of larger scale image features in the deeper layers, without needing to use larger filters or many convolutional layers. Reducing the size of the input tensor, is effectively the same as increasing the receptive field of all subsequent convolutional filters. The effective receptive field is also increased by using multiple consecutive convolutional layers. For instance, two 3x3 convolutional filters have the same effective receptive field as one 5x5 filter. However, the effective receptive field scales only linearly with the number of convolutional layers, while it scales exponentially in the case of pooling.

The pooling layers explained so far are called local pooling layers, because they pool a statistic (the maximum or the average) over a local neighborhood. However, there are also global pooling layers that pool a statistic over the whole feature map, thereby reducing each featuremap to a single value. Since a few years, it has become common practice to use a global pooling layer right before the fully-connected layers at the very end of the network.

Over the last couple of sections, we explained the fully-connected, the convolutional and the pooling layer. In the next section we will see how these layers can be stacked to form a whole convolutional neural network architecture.

ARCHITECTURE OF CONVOLUTIONAL NEURAL NETWORKS

The convolutional, pooling and fully-connected layers are the core building blocks of convolutional neural networks. Like a fully-connected neural network, a CNN is a type

of feedforward network which can be written as a composite function of these high-level building blocks. The output of the previous layer is fed-forward as input for the next layer, in this way a CNN gradually transforms the input image into the desired output.

The most common form of a CNN architecture stacks a few convolutional layers, follows them with local pooling layers, and repeats this pattern until the tensor has been reduced to a small spatial size [2]. At some point, it is common to use a global pooling layer to make the transition to fully-connected layers of which there are usually only one or two. The last fully-connected layer has no activation function. It outputs the regressed values in the case of a regression problem, and the distances to the decision boundaries in the case of a classification problem. A final softmax layer normalizes these distances into proper class probabilities that sum to 1. The class that holds the heighest probability is then selected as the predicted class.

The fully-connected layers are sometimes regarded as the actual classifier and the layers before them as the feature extraction part. The fully-connected layers at end of the CNN constitute a normal fully-connected neural network. Although fully-connected neural networks do not work when applied directly to images (see section 3.5), they do work when features have been extracted from the image. The key difference between the feature extraction in CNNs and feature extraction in other machine learning algorithms, is that the feature extraction is learned and not handcrafted. This makes CNNs very poweful, since they optimize (learn) the whole processing pipeline from the raw input till the desired output.

So far, there has only been talked about what neural network are, and how they are composed of smaller building blocks. In the next section we will see how gradient descent optimization can be used to train the free parameters in neural networks.

3.7. TRAINING NEURAL NETWORKS

In the previous chapter we have seen that the training of a machine learning algorithm is just an optimization problem and that all the theory from that subfield of mathematics applies. However, in the case of neural networks we are not really spoiled for choice. The fact that neural networks often contain millions of parameters and the fact that the datasets often consist of thousands of images, makes the usage of (quasi) second-order optimization methods infeasible [3]. Instead, it is only feasible to train neural networks using a first-order optimization method called: *Batch Stochastic Gradient Descent*.

Let us first take a look at normal gradient descent. Gradient descent is an iterative op-

[2] This traditional stacking pattern is currently being challenged by several architectures, for example: ResNets (Microsoft), Inception (Google) and Densenets (Facebook).

[3] The computation and inversion of the Hessian matrix in second-order methods, is infeasible for problems that involve so many parameters. Quasi second-order methods that approximate the inverted Hessian like BFGS, are unstable if not computed over the whole dataset. So, quasi second-order methods are also not practical for large datasets.

timizer that starts with a random parameter values and iteratively updates them until they have converged to an optimum. An iteration starts with computing the predictions on all examples, it then computes the error of these predictions, then it computes the gradient of this error with respect to the parameters, and finally it updates the parameters proportionally to this gradient, after which the iteration starts all over again with the new parameter values. This is repeated until the error has converged to a stable value. The parameter updates are proportional to the gradient of the error, since it points in the direction of the closest local minimum.

Stochastic batch gradient is a version of gradient descent that can be used to speed up convergence in the case of large datasets with many examples. It can only be used if the error over the total dataset, is expressed as the sum of error on the individual examples. Instead of computing the gradient over all examples in the dataset, it computes the gradient over a small batch of examples. This batch is usually much smaller than the total dataset. If the batch size is large enough (usually 256 examples at most), then the gradient over batch should approximately be the same as the gradient over the total dataset. Computing the gradient over a batch instead of over the whole dataset, allows for much more frequent parameter updates. These parameter updates have a stochastic nature. Meaning that the error will not strictly decrease with every update, but that it will only decrease on average. In the end this will still lead to a faster convergence, since we can make many more parameter updates (usually three orders of magnitudes) in the same time as normal gradient descent.

$$\vec{\theta} = \vec{\theta} - \eta \nabla_{\vec{\theta}} J \tag{3.7}$$

Equation 3.7 shows the gradient descent update-rule in its plain form. The $\nabla_{\vec{\theta}} J$ denotes the gradient of the error with respect to the parameters $\vec{\theta}$ and η stands for the learning rate. The learning rate is an important setting that determines by how much a parameter can change in a single update. Therefore the learning rate is sometimes also called the stepsize. Using a learning rate that is too high may result in overstepping the minimum, while using a learning rate that is too low might require a large number of iterations to converge. It is therefore common to use a learning-rate schedule in which one starts with a higher learning-rate and lowers it a few times during the training.

There are quite some variants on the vanilla update-rule in equation 3.7, that all aim to speed up convergence (achieving a lower error with fewer updates). The most common ones are *Momentum*, *Nesterovs-momentum*, *AdaDelta*, *AdaGrad*, *Adam* and *RMSprop*. Their precise form is not relevant for the work in this report, but they all try to extrapolate the error descent by keeping track of previous parameter updates. All of these variants appear regularly in literature, but the most common variants are momentum, nesterovs-momentum and adam.

Another way of speeding up convergence is called *Batch Normalization* [54]. Batch normalization replaces the bias offset term in convolutional and fully-connected layers by a function that normalizes all its input values (over the whole batch) to a standard nor-

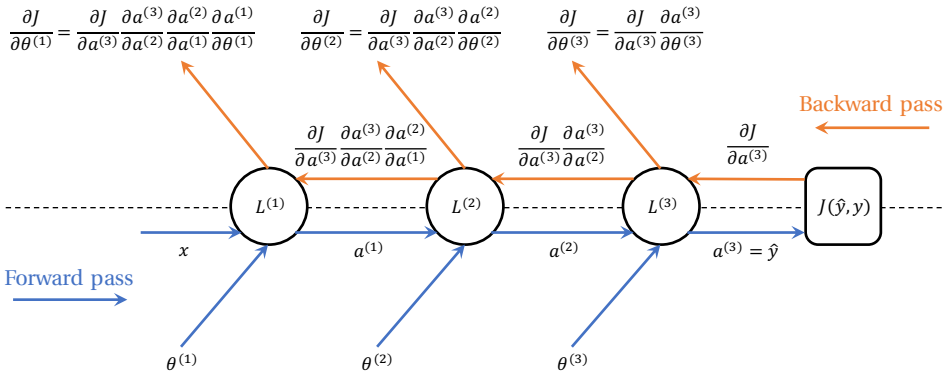


Figure 3.6: A schematic of the backpropagation algorithm for computing gradients. The forward pass first computes the predictions $\hat{y}$ and the error $J(\hat{y}, y)$ of these predictions. The backward pass propagates the error signal in reverse order through the network. Notice how the chain rule for differentiation adds an extra partial derivative term for every arrow in this schematic. At every layer, the error signal is used to compute the gradient for the parameters in that layer.

mal distribution by subtracting the mean and dividing by the standard deviation. It then multiplies the normalized values by a learnable multiplier, and adds a learnable bias offset before outputting the signal to the activation function. The exact reason why the normalization speeds up convergence is quite technical. Let us simplify by saying, that it brings the partial derivative with respect to each parameter roughly in the same order of magnitude. Because of this, batch normalization also makes the convergence less sensitive to the way in which the parameters are initialized.

EFFICIENT GRADIENT COMPUTATION WITH ERROR BACKPROPAGATION

The composite structure of feedforward networks, allows for efficient computation of the derivatives of the error with respect to the free parameters in the network. This effecient computation method is called *Error Backpropagation* or simply *Backpropagation*.

The composite structure of feedforward networks, allows us to express the partial derivative of parameters in a certain layer, as a product of the partial derivatives of the deeper layers via the chain rule for differentiation. By computing the parameter derivatives in backwards order (from the last to first layer), we can reuse large parts of the parameter derivatives in the deeper layers for the lower layers. This makes the backpropagation algorithm very efficient.

The part that can be reused is called the error signal and it is the derivative of the error with respect to the output of a particular layer. The error signal in the current layer is backpropagated to a lower layer, by multiplying it with the derivative of the layer output with respect to the layer input (chain rule). From the error signal it is only a small step to compute the parameter derivatives. The parameter derivatives in each layer, can be

computed by multiplying the error signal with the derivative of the layer output with respect to the layer parameters (chain rule).

Figure 3.6 shows schematically how the backpropagation works in practice. The forward pass first computes the predictions $\hat{y}$ and the error $J(\hat{y}, y)$ of these predictions. The backward pass propagates the error signal in reverse order through the network. Notice how the chain rule for differentiation adds an extra partial derivative term for every arrow in this schematic. The backpropagation algorithm is efficient, because it only needs to compute this extra term.

3.8. TECHNIQUES AND TRICKS TO REDUCE OVERFITTING

We already mentioned a couple of times that machine learning algorithms and neural networks in particular, suffer from the problem of overfitting. They easily achieve a very high score on the training set and then often achieve a far lower score on the test set. If this happens, one has fitted a decision boundary that is likely to be closer to the one on right of figure 2.2, than the one in the middle. Fortunately, there are some techniques and tricks that one could apply during the training stage that reduce overfitting. Since these techniques are crucial for the succes of neural networks and because some of them will be used in this research, we will mention them in this section.

The easiest way to reduce overfitting, is to collect more training examples. If the training examples would cover every region of the feature space, then the decision boundary is less likely to overfit. However, collecting more training examples is not always possible. Fortunately, data augmentation is also a good way to reduce overfitting. Data augmentation is a technique that generates new training examples from existing ones. It works by applying random distortions to existing examples, that do not change the underlying class label. For example, a horizontally mirrored image of a boat is still a valid image of a boat, but it is a whole new example for a machine learning algorithm. For images the most common distortions are: horizontal flipping, translation, rotation, scaling or a combination of these transformations. However for problems that do not involve image data, these distortions may not be so obvious and data augmentation might not be so easy to apply.

Another technique to reduce overfitting, is weight regularization. It works by adding an extra penalty to the error function for using large parameter values. This extra penalty term dynamically restricts parameter values. Meaning that the neural network can not use its entire representational power (large parameter values) to get every individiual outlier predicted correctly, but it can if it gets an entire group of examples predicted correctly. This sounds great, but there is a delicate balance between the amount of parameter restriction and the amount of correct classification. This balance is controlled by a hyperparameter that needs to be cross-validated.

Ensemble averaging is another powerful but expensive technique. It combines the predictions of multiple neural networks into a single prediction. For classification problems this could be seen as averaging the decision boundaries of multiple neural net-

works. Taking the average, removes the pecularities of individual decision boundaries while preserving the common trend. This leads to less overfitting, but evaluating a large ensemble is very expensive at test time! Dropout is a technique, that basically moves this time expense from the test stage to training stage. Only a single neural network is trained, but for each training example some neurons are randomly dropped (their output is set to zero) with a probability p. During the test stage the full neural network with all neurons is used, which basically has the same effect as averaging the outcome over multiple neural networks. However, the averaging is now implicit in the network architecture, instead of explicit over multiple architectures. It is common to apply dropout to fully-connected layers, and sometimes to convolutional layers as well. Dropout is a powerful technique, but the probability p is another hyperparameter that needs to be cross-validated.

3.9. ARCHITECTURAL DESIGN CHOICES AND HYPERPARAMETERS

The weights in the fully-connected layers and the filter weights in the convolutional layers, are optimized by the gradient descent optimizer. However finding the optimal relationship also requires making specific design choices for the CNN architecture. Examples of such architectural design choices are: the number of filters, the size of the filters, the number of convolutional layers, at which point to include subsampling layers. These design choices are called hyperparameters. It are parameters that are set before the start of the learning process. Hyperparameter optimization is usually not possible with gradient decent, because they do not have a well-defined derivative to the error. Instead, one has to train multiple times with different values to see what works best. Since the search-space grows exponentially with the number of hyperparameters, selecting these values often comes down to following heuristics [4].

In this research we propose a new type of convolutional filter of which the filter size can be optimized during the training stage. A well-defined derivative of the error with respect to the filter size, is created by imposing a mathematical structure on the filter. In the next chapter, we will explain how this new type of convolutional filter works.

[4] My personal recipe is to start with a very complex network and reduce the complexity until it is just barely capable fitting the training set perfectly. Now that you are guarantueed to have a model that is capable of learning the relationship of interest, reduce the overfitting on the training set with techniques such as: weight regularization, data augmentation, dropout, early stopping, etc. This will likely cause the performance on the training set to drop and the performance on the test set to rise. Stop when the two match or when the gap is as close as possible.

II

EXPERIMENTAL METHODS

4

STRUCTURED RECEPTIVE FIELDS

Structure is the key to a successful learning environment.

Sten Goes

THIS chapter introduces a new type of convolutional filter called a *Structured Receptive Field*. The filter size of a structured receptive field is, in contrast to normal CNN filters, variable and can be learned during the training stage. This chapter describes how the filter size of structured receptive fields can be learned, which other differences there are compared to normal CNN filters, and how structured receptive fields can be implemented in an efficient manner in a convolutional layer.

The chapter starts by explaining what structured receptive fields are and how they are motivated by scale-space theory. In section 4.2, we will see that structured receptive fields are parameterized filters and that the parameters implicitly define the effective filter weights. Section 4.3 explains that one of these parameters is a continuous scale parameter, and how optimizing this parameter can serve as proxy for optimizing the filter size. The consequences of using parametric filters instead of normal CNN filters, are discussed in section 4.4 and 4.5. We will see that the parametric model enables the learning of the filter size, but also that the resulting filters are less universal than normal CNN filters. Section 4.6 and 4.7 discuss three equivalent but different implementations of structured receptive fields in a convolutional layer. By exploiting the mathematical structure of structured receptive fields an implementation is obtained, that is significantly faster than the normal implementation. The final sections feature a complete mathematical description of structured receptive fields. We will see that structured receptive fields are expressed in terms of gaussian derivatives, and that they can be discretized and normalized in different ways. The advantages and disadvantages of the different possibilities will be discussed, before continuing to the next chapter in which the practical potential of structured receptive fields will be demonstrated with a proof of concept experiment.

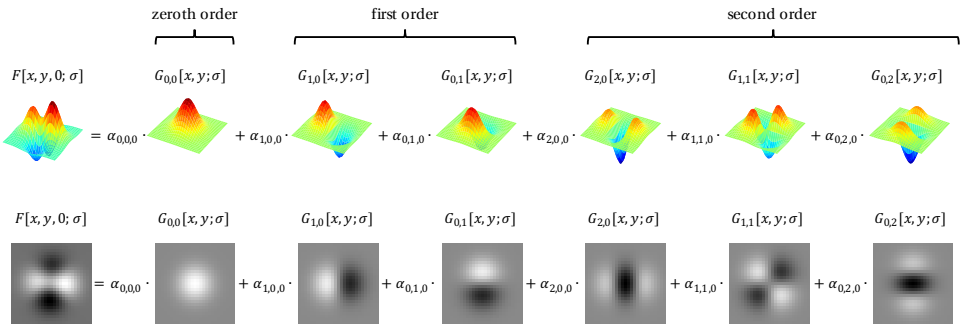


Figure 4.1: The construction of a 1-channel structured receptive field. The structured receptive field (left hand side), is a weighted sum of gaussian derivatives (right hand side). In this schematic a special notation has been used to abbreviate the notation for the gaussian derivatives. $G_{i,j}$ means i^{th} order x-derivative and j^{th} order y-derivative. The feature itself is completely defined by the weights of the linear combination (the alpha parameters) and the scale is defined by the standard deviation of the gaussian derivatives (the sigma parameter). The top row displays the effective filter and gaussian basis filters as 3-dimensional surfaces, while the bottom row shows an image in which the color intensity represents the height of these surfaces. The structured receptive field in this example contains all derivatives up to and including order two, however this is just an example. The structured receptive field can be truncated at any arbitrary order.

4.1. GAUSSIAN SCALE-SPACE

In this report we will propose a new type of filter of which the filter size can explicitly be optimized during the training stage. The basic idea for this new filter comes from scale-space theory in which the scale of image features is studied by parameterizing them with a continuous scale parameter. Thereby decoupeling the image feature itself from the scale at which it occurs in an image.

In CNNs, convolutional filters are used to detect the presence (or abscence) of image features. In scale-space theory, a filter is parameteric function with a continuous scale parameter. It can changes the size of the filter while keeping its spatial structure intact. In mathematics this property is called self-similarity. There are several mathematical functions that contain a parameter with this property, but the most important one is the *Gaussian*. The gaussian can change the size of its appearance in a continuous manner, by changing it the value of its standard deviation. Like the *Gaussian Scale-Space*, we will use gaussian derivatives to describe image features.

4.2. STRUCTURED RECEPTIVE FIELDS

The filter that we propose in this report, describes an image feature as a weighted sum of all 2-dimensional gaussian derivatives up to a certain order, see equation 4.1. In this equation, the spatial structure of the feature is completely defined by the weights of the linear combination (the alpha parameters) and the scale is defined by the standard deviation of the gaussian derivatives (the sigma parameter). Because the filters are struc-

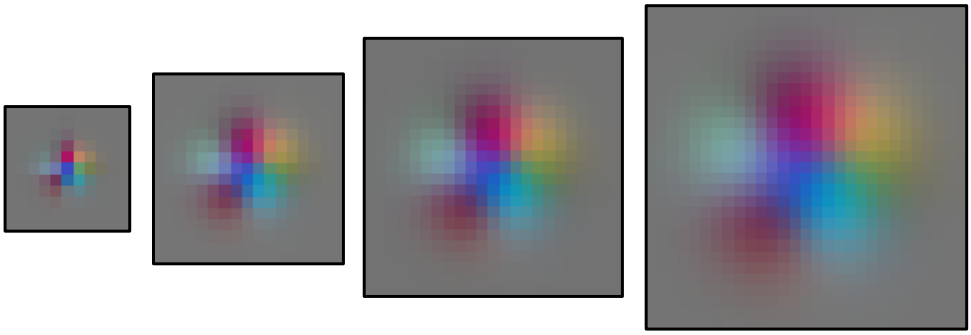


Figure 4.2: Four examples of a 3-channel structure receptive field with the same structure, but at different scales. Do not get fooled by the discrete appearance of the scales in this figure. The scale is a continuously valued parameter and for these examples it is sampled at four different values.

tured by a set gaussian derivatives of which is shown that they can accurately model the response of receptive fields in the visual cortex [30, 31], we will call these new filters *Structured Receptive Fields*.

$$F[x, y, c; \sigma] = \sum_{\substack{i \le O,\ j \le O \\ i+j \le O}} \alpha_{i,j,c} \frac{\partial^{i+j}}{\partial x^i \partial y^j} G[x, y; \sigma] \tag{4.1}$$

In the previous chapter we have seen that the filters in CNNs always extend along all channels of the image on which they operate. Notice that the same is true for structured receptive fields and that the sigma parameter and the gaussian derivatives are the same for all channels, but that the corresponding alpha parameters are not. In other words, the structured receptive fields share the same (gaussian) basis filters over the different channels.

Figure 4.1 shows the construction of a 1-channel structured receptive field. Convolutional filters with one or three channels can be visualized by grayscale and rgb-color images, respectively. For other numbers of channels the way of visualizing the filter is not so obvious. Figure 4.2 shows four examples of a 3-channel structured receptive field with the same structure, but at different scales. It clearly demonstrates the effect that changing the scale parameter has on the structured receptive field.

4.3. LEARNING THE FILTER SIZE

Structured receptive fields have the alphas and sigma as free parameters, whereas a normal CNN filters have a free parameter for each filter weight. A normal CNN filter is therefore more universal than a structured receptive field. So what is gained by imposing a mathematical structure on the convolutional filters? The continuous scale parameter provides us with the abilitiy to learn the size of the filter. For normal filters, the filter

size is a fixed hyperparameter that needs to be selected before the training stage. For structured receptive fields, the filter size is a variable that can be optimized during the training stage.

For normal CNN filters it is not possible to optimize the filter size, because it is a discrete valued parameter. Therefore it has no well-defined derivative to the error and cannot be optimized with gradient descent. The sigma parameter in structured receptive fields on the other hand, is continuously valued and has a well-defined derivative to the effective filter and therefore to the error. The chain-rule for differentiation allows to express the derivative of the error with respect to sigma as the product of two terms, see equation 4.2. The first term is the derivative of the error with respect to filter and is found by error-backpropagation just like the filters in normal CNNs. The second term is the derivative of the filter with respect to sigma and can be found by differentiating equation 4.1.

$$\frac{\partial J}{\partial \sigma} = \frac{\partial J}{\partial F} \cdot \frac{\partial F}{\partial \sigma} \tag{4.2}$$

In this report, we will optimize the alphas and the sigma during the training stage. The gaussian basis filters will be truncated at three sigmas from their center, because at that point they already contain ~99.7% of their total volume. In this way, we can use the optimization of the continuous sigma parameter as a proxy for optimizing the discrete filter size.

$$k(\sigma) = 2 \lceil 3\sigma \rceil + 1 \tag{4.3}$$

Equation 4.3 describes the truncation procedure more formally. It shows how the discrete filter size k is related to the continuous sigma. Note that there is always a well-defined center pixel, because in this definition k is always odd. This center pixel is the origin of the coordinate system of the gaussian basis filters and the filter size is the same in x- and y-direction.

4.4. CONSEQUENCES FOR THE REPRESENTATIONAL POWER

An advantage of describing the filters as a linear combination of gaussian derivatives is that it provides us with a natural way to learn filter sizes. The downside is that the set of gaussian basis filters spans only a small subset of all possible filters that a normal CNN filter can learn. However, this loss in universality should not be a problem, if the subset spanned by the gaussian derivatives contains all filters that we would like to learn [1].

Fortunately, there is a good reason to believe that structured receptive fields are capable of approximating every filter that is of interest. Appendix A shows that a structured

[1] The fact the normal filters could learn every possible weight configuration, does not mean that they do. For example, the learned filters in the first layer of normal CNN usually look alot like gabor and gaussian derivative filters. So, these filters can also be learned by structured receptive fields.

receptive field can be rewritten in such a way, that it resembles a Taylor expansion that is modulated by a gaussian envelope. Therefore any normal filter can be approximated in the center region by the Taylor expansion, while the gaussian envelope supresses all the values to zero near the border. This suppression near the border should be desirable, because it is inline with the principle of local connectivity. This principle was one of the main motivations for using convolutional network and states that the statistical dependency between pixels falls off as distance increases. Structured receptive fields capture the essence of local connectivity, by explicitly modulating filters with a gaussian envelope.

Although these arguments might seem plausible, it is still true that a normal filter (that is large enough) can resemble any structured receptive field, while the reverse is not always possible. The representational power of structured receptive field is therefore something that needs a careful investigation and we will do so with an experiment in section 5.1.

4.5. REGULARIZING PROPERTIES

Structured receptive fields have two properties that could make them easier to regularize than normal filters. The first property, is that one can explicitly limit the complexity of the filter by truncating the linear combination at a certain order. This truncation of the Taylor expansion should have a regularizing effect since including higher order derivatives gives the filter extra ability to learn very noisy structures. The second property is that structured receptive fields can increase their filter size without needing to add more parameters. This should also have a regularizing effect, especially when the filters are large [32].

4.6. THE STRUCTURED CONVOLUTIONAL LAYER

This section describes the algorithmic implementation of structured receptive fields in convolutional layers. We will see that significant speed ups can be achieved by sharing parts of computations between the different feature maps, and that for large filters another big speedup is gained by performing the convolutions in the frequency domain. Since these performance optimizations are only possible or relevant for structured receptive fields, we will refer to this layer as a *Structured Convolutional Layer*.

$$a^{(l)}[x,y,n] = \phi\left(\theta_0^n + \sum_{f_x=0}^{k_x-1}\sum_{f_y=0}^{k_y-1}\sum_{c}^{C} a^{(l-1)}[x+f_x, y+f_y, c] \cdot F_n[f_x, f_y, c]\right) \tag{4.4}$$

In a normal convolutional layer, the computation of a single output neuron at location x, y in the n^{th} featuremap is given by equation 4.4 (see chapter 3). By adopting a special notation for the cross-correlation operation, we can drop the spatial coordinates and conveniently express the computation of the entire n^{th} feature map $a^{(l)}[n]$ in terms of

the channels of the input tensor $a^{(l-1)}[c]$ and the corresponding filter channels $F_n[c]$, see equation 4.5.

$$a^{(l)}[n] = \phi\left(\theta_0^n + \sum_c^C a^{(l-1)}[c] \star F_n[c]\right) \tag{4.5}$$

where $\star$ stands for cross-correlation.

By far the largest amount of computations are needed to evaluate the cross correlations terms in this equation. Fortunatly, one can obtain different expressions that are equivalent mathematically, but require far less computations to evaluate algorithmically. Equation 4.6 shows the expressions that describe the three different implementations that will be investigated in this report. From top to bottom, we will call them the *Normal implementation*, the *Fast implementation* and the *Faster implementation*.

$$\sum_c^C a^{(l-1)}[c] \star F_n[c] = \sum_c^C a^{(l-1)}[c] \star \left(\sum_{i,j}^M \alpha_{i,j,c,n} \frac{\partial^{i+j} G}{\partial x^i \partial y^j}\right) \tag{4.6a}$$

$$= \sum_c^C \sum_{i,j}^M \alpha_{i,j,c,n} \left(a^{(l-1)}[c] \star \frac{\partial^{i+j} G}{\partial x^i \partial y^j}\right) \tag{4.6b}$$

$$= \mathscr{F}^{-1}\left\{\sum_c^C \sum_{i,j}^M \alpha_{i,j,c,n} \left(\mathscr{F}\left\{a^{(l-1)}[c]\right\}^* \cdot \mathscr{F}\left\{\frac{\partial^{i+j} G}{\partial x^i \partial y^j}\right\}\right)\right\} \tag{4.6c}$$

where $\mathscr{F}\{\cdot\}$ and $\mathscr{F}^{-1}\{\cdot\}$ denote the forward and backward Fourier transform.

Figure 4.3 supports the following explanations of how the different implementations work and how they achieve their speedup.

The normal implementation is shown on the right hand side of equation 4.6a. It is obtained by substituting the expression for structured receptive fields (equation 4.1), into equation 4.5. In this implementation the effective filters are computed first, then each channel of the effective filter is correlated with the corresponding input channel after which the results are summed to create the feature map. Apart from the step where the effective filters are computed, this just a normal convolutional layer. Hence the name, normal implementation.

The fast implementation is shown in equation 4.6b. It is obtained from equation 4.6a, by using the linearity property of the cross correlation. This implementation first correlates each input channel with every gaussian basis filter. Because the result does not depend on a specific featuremap (no dependency on n), it can be reused in the computation

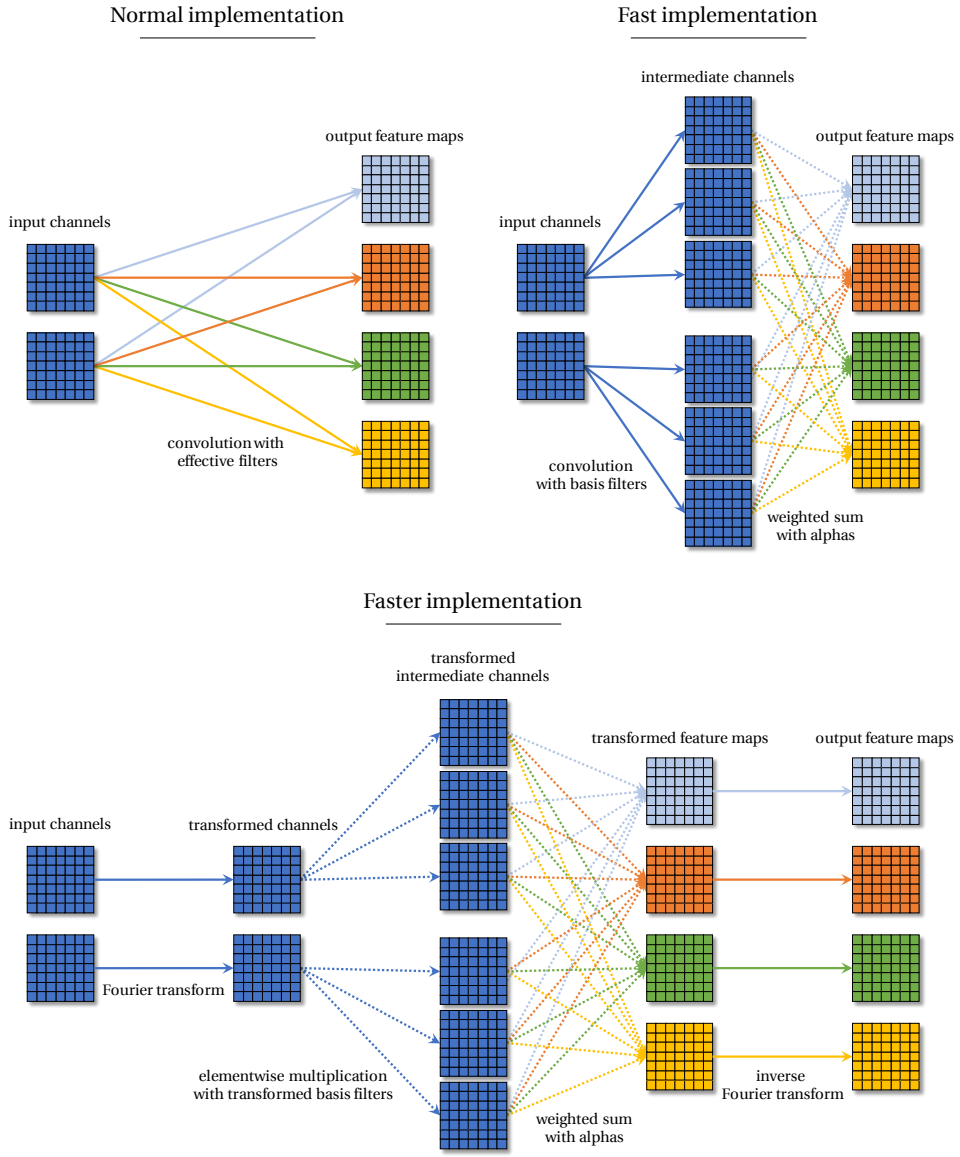


Figure 4.3: Three different but equivalent implementations of a structured convolution layer. For the purpose of this illustration, it is more convenient to use a 2D visualization. Therefore the depth slices of the 3D tensors (i.e. input channels) are shown below each other. The arrows in this schematic stand for mathematical operations. An arrow with a dark blue color indicates that the operation is shared over all featuremaps. Other colors indicate feature map specific operations. Solid arrows indicate operations that are much more expensive to compute than the operations indicated by dashed arrows in the same schematic. So roughly speaking, one would like to minimize the number of solid arrows.

of other feature maps. The feature map is computed by taking a feature map specific weighted sum of the shared intermediate channels. The fast implementation changes the number of cross-correlations from $C \cdot N$ to $C \cdot M$. Thereby it takes advantage of the fact that the number of basis filters M is usually far lower than the number of feature maps N.

Finally, the faster implementation is shown in equation 4.6c. It is obtained from equation 4.6b, by using the correlation theorem to perform the correlation in the frequency domain and by using the linearity of the fourier transform to move the summation inside the inverse transform. This implementation first transforms each input channel and conjugates them, then each transformed channel is elementwise multiplied by every transformed basis filter. This intermediate result is again independent of n and therefore can be reused in the computation of other feature maps. Next, a transformed feature map is created by taking a feature map specific weighted sum after which it is transformed back to the spatial domain by the inverse fourier transform, to arrive at the output feature map.

The faster implementation reduces the number of cross-correlations to zero, but it adds a number of fourier transforms instead. When the filters are large these fourier transforms are relatively cheap to compute, compared to the cross-correlations. Hence, this implementation will be a lot faster when the filter sizes are large. An important advantage of the gaussian basis filters is that they do not have to be evaluated in the spatial domain, before being transformed to the frequency domain. Instead they can be evaluated directly in the frequency domain, because they have known analytical expressions in both domains.

This section described the three different implementations. It explained in which scenarios one might be faster than the other. The next section will compare the different implementations in a more quantative manner.

4.7. PERFORMANCE OF DIFFERENT IMPLEMENTATIONS

It is common to compare the relative performance between different algorithms, by counting the number of elementary operations needed to evaluate the outcome. The number of elementary operations is often expressed in terms of the relevant parameters, such that one can easily compare the algorithms for different scenarios. This type of analysis is called time-complexity analysis, since it gives a rough estimate for the running time of an algorithm.

We will count the number of multiplications needed for each implementation, and we will express it in terms of the relevant parameters: the spatial size of the input tensor S, the number of input channels C, the number of feature maps N, the number of basis filters M, the filter size k. With these parameters, cross-correlation of a single input channel with a single filter channel requires S^2k^2 multiplications. The 2D fast fourier transform (FFT) requires $S^2 \log S^2$ multiplications. However the input tensor first needs

Normal:	$CMNk^2+$	computing effective filters
	CNS^2k^2	correlation with effective filters
Fast:	CMS^2k^2+	correlation with basis filters
	$CMNS^2$	weighted sum with alphas
Faster:	$C(S+k)^2\log(S+k)^2+$	fourier transform of input channels
	$CM(S+k)^2+$	multiplication with transformed basis filters
	$CMN(S+k)^2+$	weighted sum with alphas
	$N(S+k)^2\log(S+k)^2$	inverse fourier transform

Table 4.1: Performance benchmarks for the different implementations for common settings of the relevant parameters. Most of these scenarios occur in DenseNet architecture, which is the neural network architecture that currently holds the state-of-the-art results on several datasets. The DenseNet architecture will be used for most of the experiments in chapter 5.

S	C	N	M	k	Normal	Fast	Faster
32	76	12	6	11	113M	62M	12M
16	148	12	6	13	78M	41M	11M
8	220	12	6	25	115M	53M	21M
32	76	12	15	11	114M	155M	29M
16	148	12	15	13	81M	102M	25M
8	220	12	15	25	130M	134M	49M

to be zero-padded to a spatial size of $S+k$, in order to turn the circular correlation into a linear correlation.

The calculations below show the amount of floating point multiplications (flops) that are needed in each implementation. Since it is not easy to compare the implementations from these expressions, we evaluated them for a few common scenarios in table 4.1. Most of these scenarios occur in the DenseNet architecture, which is the neural network architecture that currently holds the state-of-the-art results on several datasets.

4.8. CREATING THE GAUSSIAN BASIS FILTERS

This section features the construction of the gaussian basis filters. Since gaussian derivatives have a continuous definition, the question arises how to obtain discrete filters? The scale-space theory for continuous signals, elegantly derived the gaussian function as the only function satisfying all the scale-space axioms (a set of desirable properties, see section 1.5). However images are discrete rather than continuous signals, and obtaining a

discrete filter by simply sampling a continuous gaussian derivative will cause the resulting filter to no longer satisfy the scale-space axioms.

Fortunatly, scale-space theory also contains a modification for discrete signals [55]. This modification defines discrete analogs for the continuous gaussian derivatives. These discrete analog derivatives are defined in such a way that all scale-space properties that hold for continuous signals transfer to the discrete signals.
They are obtained by replacing:

- the continuous gaussian kernel by a discrete analog of the gaussian kernel,
- and the derivative operators with a set of difference operators.

For convienience, we will call the discrete analog derivatives simply *discrete gaussian derivatives* and the sampled continuous derivatives simply *continuous gaussian derivatives*. The exact definition of the discrete gaussian derivatives will be given later in this section.

Using discrete derivatives instead of continuous derivatives will be mostly theoretical, at large scales where the grid effects can be expected to be small. However at small scales and for high order derivatives these grid effects are not so small and the resulting discretization errors could be substantial. Hence, our main choice will be to use discrete derivatives. Nevertheless, we will also implement the continuous derivatives as a sanity check and we will compare both methods and their effect on learning the sigma parameter in section 5.2 with an experiment.

THE CONTINUOUS AND THE DISCRETE GAUSSIAN DERIVATIVES

The continuous gaussian is the Green's function that solves the continuous time diffusion equation on a continuous spatial grid. Analogously, the discrete gaussian is defined as the Green's function that solves the continuous time diffusion equation on a discrete spatial grid. A Green function describes how the spatial initial solution of a partial differential equation evolves over time, by expressing solution as the convolution of the initial solution with a time dependent Green's function. The discrete gaussian turns out to be an exponentially weighted modified bessel function of the first kind, in which the order is the spatial coordinate and the argument is the variance. Equation 4.7 shows both the continuous gaussian (left) and the discrete gaussian (right).

$$G(x;\,\sigma) \;=\; \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{1}{2}\frac{x^2}{\sigma^2}}, \qquad G[x;\,\sigma] \;=\; e^{-\sigma^2}\, I[x;\,\sigma^2] \tag{4.7}$$

To avoid confusion, the continuous gaussian will be denoted with round brackets $G(\cdot)$ and the discrete gaussian will be denoted with square brackets $G[\cdot]$. In two dimensions, both the continuous and discrete gaussian are seperable. Meaning that the 2-dimensional gaussian can be written as the product of the two 1-dimensional gaussians, and that the 2-dimensional derivatives can be written as the product of two 1-dimensional derivatives. See equation 4.8.

Table 4.2: The central difference coefficients of accuracy two, up to order six. For higher orders, the coefficients can be found by the recurrence relation in equation 4.11.

order	-3	-2	-1	0	1	2	3
0				1			
1			$-\frac{1}{2}$	0	$\frac{1}{2}$		
2			1	-2	1		
3		$-\frac{1}{2}$	1	0	-1	$\frac{1}{2}$	
4		1	-4	6	-4	1	
5	$-\frac{1}{2}$	2	$-\frac{5}{2}$	0	$\frac{5}{2}$	-2	$\frac{1}{2}$
6	1	-6	15	-20	15	-6	1

$$G[x,y;\,\sigma] = G[x;\,\sigma]\;G[y;\,\sigma], \qquad \frac{\partial^{i+j} G[x,y;\,\sigma]}{\partial x^i \partial y^j} = \frac{\partial^i G[x;\,\sigma]}{\partial x^i}\;\frac{\partial^j G[y;\,\sigma]}{\partial y^j} \tag{4.8}$$

Equation 4.9 show that a continuous gaussian derivative of arbitrary order i can be expressed in terms of a hermite polynomial $H_i(x)$, whereas a discrete gaussian derivative of arbitrary order i is expressed in terms of a finite difference operator δ_{x^i} that operates on the discrete gaussian.

$$\frac{\partial^i G(x;\,\sigma)}{\partial x^i} = \left(\frac{-1}{\sqrt{2}\sigma}\right)^i H_i\left(\frac{x}{\sqrt{2}\sigma}\right) G(x;\,\sigma), \qquad \frac{\partial^i G[x;\,\sigma]}{\partial x^i} = \delta_{x^i} G[x;\,\sigma] \tag{4.9}$$

The hermite polynomials and the finite difference operators, are both defined in terms of a recurrence relations, see equation 4.10 and 4.11.

$$\begin{aligned} H_i(x) &= 2x\,H_{i-1}(x) - 2(i-1)\,H_{i-2}(x) \\ H_0(x) &= 1 \\ H_1(x) &= 2x \end{aligned} \tag{4.10}$$

There exist two types of hermite polynomials, but these are the 'physicists' hermite polynomials.

$$\begin{aligned} &\delta_{x^{2i}}\,f[x] = (\delta_{x^2})^i\,f[x] \quad \text{and} \quad \delta_{x^{2i+1}}\,f[x] = (\delta_{x^2})^i \delta_{x^1}\,f[x] \\ &\delta_{x^1}\,f[x] = -\tfrac{1}{2} f[x-1] + \tfrac{1}{2} f[x+1] \\ &\delta_{x^2}\,f[x] = f[x-1] - 2f[x] + f[x+1] \end{aligned} \tag{4.11}$$

The finite difference operators in this equation are defined by a kernel of which the coefficients are the central difference coefficients of accuracy two [56]. For the purpose of illustration, all coefficients up to order six are given in table 4.2. Normally the choice of difference operators would be more or less arbitrary, because the derivative of a discrete signal is not well-defined. However these operators are chosen such that all continuous space-scale properties transfer to the discrete domain. Note that a consequence of this choice is that: $\delta_{x^1}\delta_{x^1} \neq \delta_{x^2}$ and that odd and even order derivatives follow a different recurrence relation.

The following summary contains all the expressions needed to construct the continuous and discrete gaussian derivatives in the spatial domain. Furthermore, it also includes the fourier transform of the discrete gaussian basis filters for direct evaluation in the frequency domain in case of the 'faster' implementation.

Construction of the Gaussian basis filters

A summary containing all the expressions needed to construct the continuous and discrete gaussian derivatives in the spatial or the frequency domain.

Continuous gaussian derivatives the in spatial domain

$$\frac{\partial^{i+j} G(x,y;\sigma)}{\partial x^i \partial y^j} = \frac{\partial^i G(x;\sigma)}{\partial x^i}\,\frac{\partial^j G(y;\sigma)}{\partial y^j}$$

$$\frac{\partial^i G(x;\sigma)}{\partial x^i} = \left(\frac{-1}{\sqrt{2}\sigma}\right)^i H_i\left(\frac{x}{\sqrt{2}\sigma}\right) G(x;\sigma)$$

$$H_i(x) = 2x\,H_{i-1}(x) - 2(i-1)\,H_{i-2}(x)$$

$$H_0(x) = 1$$

$$H_1(x) = 2x$$

$$G(x;\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{1}{2}\frac{x^2}{\sigma^2}}$$

$$\frac{\partial}{\partial\sigma}\frac{\partial^i G(x;\sigma)}{\partial x^i} = \left(\frac{-1}{\sqrt{2}\sigma}\right)^i \frac{1}{\sigma^3}\left[(x^2-(1+i)\sigma^2)H_i\left(\frac{x}{\sqrt{2}\sigma}\right) - \sqrt{2}ix\sigma H_{i-1}\left(\frac{x}{\sqrt{2}\sigma}\right)\right] \cdot G(x;\sigma)$$

Discrete gaussian derivatives in the spatial domain

$$\frac{\partial^{i+j} G[x,y;\,\sigma]}{\partial x^i \partial y^j} = \frac{\partial^i G[x;\,\sigma]}{\partial x^i}\,\frac{\partial^j G[y;\,\sigma]}{\partial y^j}$$

$$\frac{\partial^i G[x;\,\sigma]}{\partial x^i} = \delta_{x^i} G[x;\,\sigma]$$

$$\delta_{x^{2i}} f[x] = (\delta_{x^2})^i f[x] \quad \text{and} \quad \delta_{x^{2i+1}} f[x] = (\delta_{x^2})^i \delta_{x^1} f[x]$$

$$\delta_{x^1} f[x] = -\tfrac{1}{2} f[x-1] + \tfrac{1}{2} f[x+1]$$

$$\delta_{x^2} f[x] = f[x-1] - 2f[x] + f[x+1]$$

$$G[x;\,\sigma] = e^{-\sigma^2}\, I[x;\,\sigma^2]$$

$$\frac{\partial}{\partial\sigma}\frac{\partial^i G[x;\,\sigma]}{\partial x^i} = e^{-\sigma^2}\delta_{x^i}\left(I[x-1;\,\sigma^2] - 2\sigma I[x;\,\sigma^2] + I[x+1;\,\sigma^2]\right)$$

Discrete gaussian derivatives in the frequency domain

$$\frac{\partial^{i+j} G[\omega_x,\omega_y;\,\sigma]}{\partial x^i \partial y^j} = \frac{\partial^i G[\omega_x;\,\sigma]}{\partial x^i}\,\frac{\partial^j G[\omega_y;\,\sigma]}{\partial y^j}$$

$$\frac{\partial^i G[\omega_x;\,\sigma]}{\partial x^i} = \mathcal{F}\left\{\frac{\partial^i G[x;\,\sigma]}{\partial x^i}\right\} = G[\omega_x;\,\sigma]\sum_p \delta_{x^i}\{p\}\, e^{-ip\omega_x}$$

$\delta_{x^i}\{p\}$ denotes the coefficient corresponding to the shift p (see table 4.2).

$$G[\omega_x;\,\sigma] = \mathcal{F}\{\,G[x;\,\sigma]\,\} = e^{-\sigma^2\,[\cos(\omega_x)-1]}$$

$$\frac{\partial}{\partial\sigma}\frac{\partial^i G[\omega_x;\,\sigma]}{\partial x^i} = -2\sigma e^{-\sigma^2\,[\cos(\omega_x)-1]}\sum_p \delta_{x^i}\{p\}\, e^{-ip\omega_x}$$

4.9. NORMALIZATION OF THE GAUSSIAN BASIS FILTERS

This final section covers the normalization of the gaussian basis filters. The normalization brings the magnitude of each basis filter in approximately the same range. Figure 4.4 shows that if the basis filters are not normalized, the higher order filters are dwarfed by the by the lower order filters. Normalizing the Root Mean Square (RMS) value of each basis filter to unity, ensures that the magnitude of each basis filter is approximately in the same range.

For the respresentational power, the normalization constant is not important because it can also be absorbed in the alpha parameter. Practically however, it turns out to be

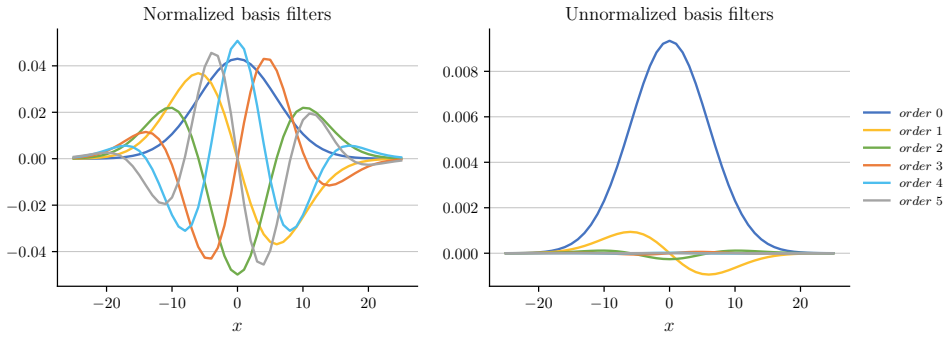


Figure 4.4: Visual comparison of normalized and unnormalized filters. For the purpose of illustration, only the x-dimension of the filter is shown. Notice that in the case of the unnormalized filters, the higher order filters are dwarfed by the by the lower order filters. Normalizing the RMS value of each basis filter to unity, ensures that the magnitude of each filter is approximately in the same range. This happens to be important for learning the alpha and sigma parameters, as we will see later in this report.

very important for learning the right values for the alpha and sigma parameters. The gradient descent optimizer works much better, if the partial derivative of each parameter are in approximately the same order of magnitude. The stepsize of the parameter update is namely proportional to the partial derivative. So if the partial derivatives have totally different magnitudes then choosing an appropriate learning rate is impossible. For a certain learning rate, some parameters will overstep the minimum while others require so many steps that convergence takes forever. Normalization of the basis filters tries to solve this problem, by bringing the alpha and sigma derivatives in the same order of magnitude regardless of the order of the structured receptive field.

There are many different ways in which one could normalize basis filters, but the exact method probably does not matter as long as the basis filters have the same magnitudes after normalization. In this report we normalize each basis filter by its RMS value, see equation 4.12. We use the RMS value, because it can be evaluated directly in the frequency domain. Via parsevals relation, it is easy to show that the RMS value in the spatial domain equals the RMS value in the frequency domain. Direct evaluation of the basis filters in the frequency domain is a necessary requirement to make full use of the 'faster' implementation, mentioned in section 4.6.

$$\frac{\partial^{i+j} G_{\text{norm}}[x,y;\sigma]}{\partial x^i \partial y^j} = \frac{\frac{\partial^{i+j} G[x,y;\sigma]}{\partial x^i \partial y^j}}{\sqrt{\frac{1}{k^2}\sum_x^k \sum_y^k |\frac{\partial^{i+j} G[x,y;\sigma]}{\partial x^i \partial y^j}|^2}} \tag{4.12}$$

In chapter 5, some of the experiments will be repeated without normalizing the basis filters to investigate the effect of normalization.

III

EXPERIMENTS AND RESULTS

5

PROOF OF CONCEPT

A proof of concept is a realization of a certain method or idea, that aims to demonstrate that some concept has practical potential.

Wikipedia

THIS chapter describes three experiments, that together aim to demonstrate the practical potential of structured receptive fields. Each experiment serves its own purpose. The first experiment is described in section 5.1 and its purpose is to investigate the representational power of structured receptive fields. Structured receptive fields must be expressed in terms of gaussian derivatives in order to be able to learn the filter size. In this experiment we question whether structured receptive fields are able to approximate the filters learned by normal CNNs, despite this imposed mathematical form.

The second experiment is described in section 5.2 and its purpose is to show that the feature scale (filter size) can be learned during the training stage. In this experiment the same network architecture is being trained twice: once on original images and once on the same but resized images. If structured receptive fields are able to learn a meaningful scale, then the ratio between the learned scales should be the same as the resize factor of the images.

Section 5.3 describes the third and most important experiment. Its purpose is to demonstrate an actual advantage when using structured receptive fields instead of normal filters. In this experiment, we resemble the network architecture that holds the state-of-the-art results on several benchmark datasets. Ceteris paribus, we swap all normal filters for structured receptive fields. If structured receptive fields have any advantage over normal filters, a higher classification accuracy on the test set should be observed.

5.1. REPRESENTATIONAL POWER

We start by investigating the representational power of structured receptive fields. The main purpose of this experiment is to see whether structured receptive fields can approximate the filters learned by normal CNNs. To this end, we download a pre-trained AlexNet and try to approximate the filters with structured receptive fields. These approximations are obtained by fitting the alphas and sigmas of the structured receptive fields against the original AlexNet filters. Visual comparison will tell whether the structured receptive fields are capable of approximating the original AlexNet filters.

ALEXNET

AlexNet is the network architecture that won the ImageNet competition in 2012 [11]. It is named after Alex Krizhevsky, who designed it. The remarkable results it achieved in the ImageNet competition (~10% higher than any other competitor), marked the beginning of the era of CNNs. Almost all of the currently used architectures are in some way derived from AlexNet.

AlexNet consists of five convolution layers followed by three fully connected layers. The 11x11 filters used in the first convolution layer of AlexNet are quite large by todays standards. Nowadays it is much more common to use only 3x3 filters. Large filters could in theory display more oscillations than smaller filters and could therefore be harder to approximate with low order gaussian derivatives. The AlexNet filters were chosen with this idea in mind that if structured receptive fields would be capable of approximating these large filters, then chances are high that they can also approximate the smaller filters of other architectures. Though, this is something that could be investigated further.

EXPERIMENTAL SETTINGS

For this experiment, we truncate the structured receptive fields at order 4. Meaning that we include all gaussian derivatives up-to and including order 4. The sigmas are initialized at 2.75 and the alphas are initialized uniform at random between -0.01 and 0.01. The final results after 1000 steps of gradient descent are shown in figure 5.1. In order not to take up all the space, we only show the first 10 filters of the first convolutional layer of the AlexNet. The other 86 filters have similar approximations and can be found in appendix B, together with the approximations to filters of the deeper convolutional layers.

RESULTS AND CONCLUSIONS

From the results in figure 5.1, it is clear that structured receptive fields have no trouble in approximating most of the original AlexNet filters. It only has trouble approximating the penultimate filter, which is understandable because it contains high frequency oscillations that cannot be modeled by low order derivatives. Gaussian derivatives have the property that with every order of derivative, an extra zero crossing is added. So, a fourth order gaussian derivative has only 4 zero crossings, which is not enough to model the penultimate filter. The approximation becomes better when the order of the structured receptive field is increased to 11, see 5.2. Though, we conclude that order 4 is enough to approximate most of the AlexNet filters.

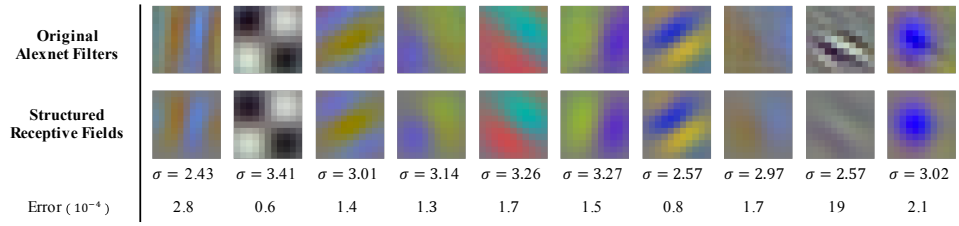


Figure 5.1: Structured receptive field approximations of AlexNet filters. The top row shows the first 10 filters with spatial dimensions of 11x11, from the first convolutional layer of the AlexNet. The second row shows the structured receptive field approximations of these AlexNet filters. This approximation was obtained by fitting the alphas and sigma of the structured receptive field against the original AlexNet filter. The final mean squared error after 1000 steps of gradient descent and the fitted sigma are shown in the third and fourth row. The structured receptive fields used for this approximation, consist of all gaussian derivatives up-to and including order 4.

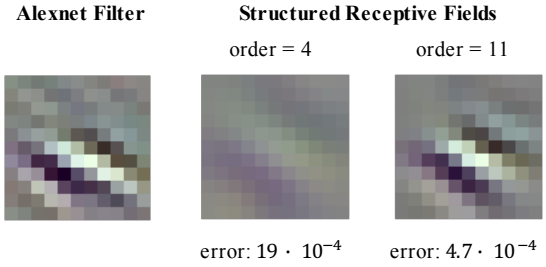


Figure 5.2: Comparing structured receptive field approximations with different truncation orders. The image on the left depicts the 9th AlexNet filter; the middle image shows a structured receptive field approximation of order 4, and the image on the right is an approximation of order 11. It is clear that increasing the order improves the approximation both visually and numerically.

It is remarkable that the structured receptive fields are capable of approximating the AlexNet filters so well, because they only contain 46 parameters (3 channels x 15 alphas + 1 sigma) whereas the original AlexNet filters contain 363 parameters (3 channels x 11x11 filters). This shows that many of the parameters in the AlexNet are redundant, and that the number of parameters can be reduced substantially, if the filters are expressed in a different basis! The method of registering images against a set of basis images to reduce the amount of parameters needed to represent the full image, looks a lot like JPEG compression. But difference is, that in JPEG compression the parameters can be found exactly and efficiently by orthogonal projection.

For performance reasons, it will be needed (at least in our current implementation) that the structured receptive fields in the same convolutional layer share the same set of gaussian derivatives. This means that the structured receptive fields must have the same or-

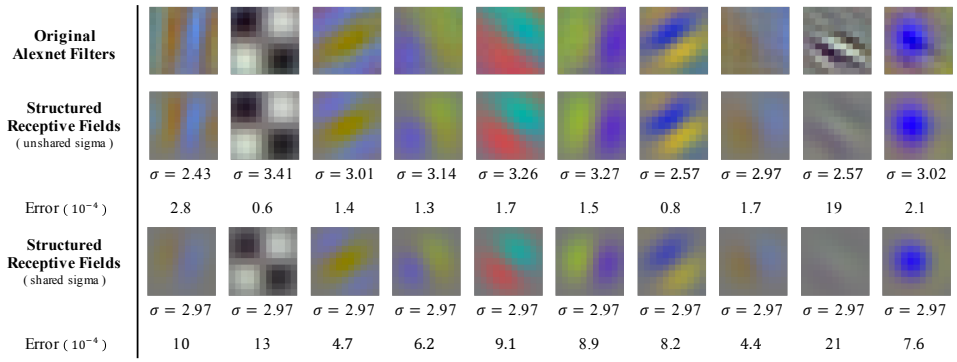


Figure 5.3: structured receptive field approximations of the first 10 AlexNet filters. The top row shows the original AlexNet filters. In the middle row each approximation has its own sigma, while in the bottom row the same sigma is shared over all approximations. As a result of sharing the sigma, the structured field must make compromises that cause the approximation to become worse than when each approximation has its own sigma.

der and share the same sigma. Therefore we will investigate, how the approximations change when all structured receptive fields are restricted to have the same sigma. The results of this can be seen in figure 5.3. A first thing to notice, is that the shared sigma exactly matches the average of the unshared sigmas. The approximations have of course become worse, but the drop in quality seems to be quite small. Though its influence on the classification performance could be investigated further. Especially, since the number of compromises that need to be made as a result of sharing the sigma, grows with the number of filters that share the same sigma. With the current hardware and implementation however, it is simply not feasible to give each filter its own sigma. Therefore shared sigma filters will be used in the remainder of this report.

A BASIC INTUITION FOR TRAINING STRUCTURED RECEPTIVE FIELDS

Besides the main purpose of investigating the representational power, there are more things that can be learned from this experiment. For instance, there are some issues regarding the usage of gradient descent optimization that we can investigate. Gradient descent optimizers are not guaranteed to find the best solution. Instead, convergence to an optimal solution relies on the initial guess of the parameters, as well as the curvature of the error landscape. It relies on whether there exists a path in the error landscape from the initial guess to the optimum along which the error only descents [1]. In general, it is desirable for gradient descent to start from an initial guess that is close to the optimum and to have an error landscape whose descent has the same order of magnitude in each direction. Fortunately, we can influence initial guess by choosing a different initialization scheme and we can influence the curvature of the error landscape by normalizing the

[1] Because parameters are updated proportional to their derivative, this descent cannot be too slow. Otherwise it will take forever to reach the optimum. This problem is called the vanishing gradient problem and occurs often in very deep neural networks.

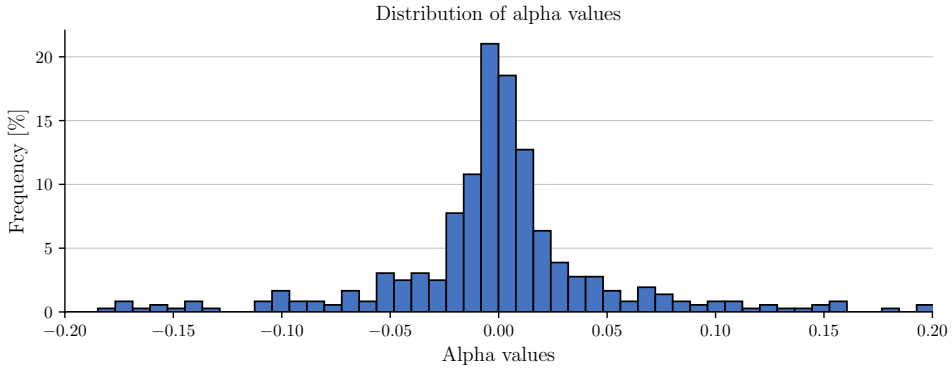


Figure 5.4: The distribution of the alphas in the structured receptive field approximations in the bottom row of figure 5.3.

gaussian derivatives. The normalization of the basis filters brings the partial derivative with respect to each alpha of structured receptive field in the same order of magnitude, see section 4.9.

To find an initial guess close to the optimum, we will study the distribution of the alpha parameters in the approximations of the shared sigma filters in figure 5.3. In figure 5.4, we see that these alphas are approximately normally distributed around zero with a standard deviation of approximately 0.02. This knowledge can be used to initialize the alphas in further experiments. Furthermore, it can be concluded that normalizing the basis filters is absolutely crucial for the structured receptive field approximations to converge. In figure 5.5, we provide two examples to convince the reader that both initialization and normalization are indeed important when using gradient descent.

The first row of this figure shows the original AlexNet filters. The second row shows the successful structured receptive fields approximations, that were found with normalized gaussian derivatives and alphas initialized uniform at random between -0.01 and 0.01. The third and fourth row shows the structured receptive field approximations failing to converge because the gaussian derivatives were not normalized or because they were initialized too far from the global optimum. The alphas for the fourth row were initialized uniform at random between -1.0 and 1.0.

5.2. LEARNING THE FEATURE SCALE

Now that a basic intuition for using structured receptive fields has been developed, it is time to embed them into CNNs. The main goal of this experiment is to investigate whether the feature scale can be learned during the training stage. This will be put to the test, by training the same CNN architecture on both normal and resized versions of images in the MNIST dataset. A comparison of learned scales on both datasets will

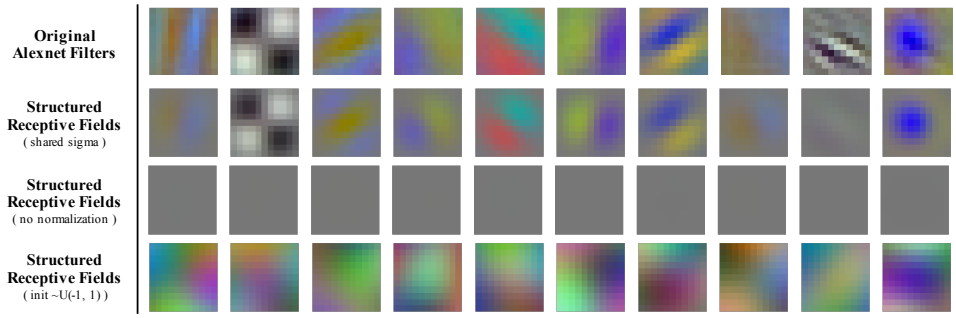


Figure 5.5: Two examples of the gradient descent optimizer failing to approximate the AlexNet filters. The first row shows the original AlexNet filters. The second row shows the successful structured receptive fields approximations, that were found with normalized gaussian derivatives and alphas initialized uniform at random between -0.01 and 0.01. The third row shows the structured receptive field approximations failing to converge because the gaussian derivatives were not normalized. In the fourth row, the structured receptive field approximations got stuck in a local minima because they were initialized too far from the global optimum. The alphas for the fourth row were initialized uniform at random between -1.0 and 1.0.

5

reveal whether structured receptive fields are capable of learning the feature scale. For example, the feature scale learned on the 2.0x enlarged images should be twice the scale learned on the normal images This can be understood by looking at the exponent of the gaussian: $e^{\frac{-(x^2+y^2)}{2\sigma^2}}$ and noting that any uniform scaling of the spatial coordinates can be canceled by scaling the standard deviation with the same factor.

THE MNIST DATASET

The MNIST dataset was created by the National Institute of Standards and Technology and consists of grayscale images of 28x28 pixels, each containing a single handwritten digit. By combining two original NIST datasets, they formed a dataset that contains 60,000 training images and 10,000 testing images [57]. The MNIST dataset was selected specifically for this experiment because all the digits appear at roughly the same scale. We create two bilinearly upsampled versions of the original dataset (1.0x MNIST) to which we will refer as 1.5x MNIST and 2.0x MNIST.

NETWORK ARCHITECTURE AND TRAINING

To avoid room for other explanations while keeping the results relevant, we use the smallest possible CNN architecture that contains all the commonly used layers (except dropout). The feature extraction part of the architecture consists of a structured convolution (without bias) followed by batch normalization, a relu activation function and max pooling layer. The classification is done with a single fully connected layer (with bias), followed by a softmax layer to convert the outcomes to proper class probabilities. The stride and size of the max pooling operation are adjusted for each dataset, to keep the dimensionality of the feature vector going into the fully connected layer equal for all three datasets. For the 1.0x MNIST a stride of 2 and a size of 2x2 is used; for the 1.5x

Table 5.1: The network architecture that is used to investigate whether structured receptive fields can learn the feature scale. The stride and size of the max pooling operation are adjusted for each dataset, to keep the dimensionality of the feature vector going into the fully connected layer equal for all three datasets. For the 1.0x MNIST a stride of 2 and a size of 2x2 is used; for the 1.5x MNIST a stride of 3 and a size of 3x3 is used and for the 2.0x MNIST a stride of 4 and a size of 4x4 is used.

MNIST image		
bilinear upsampling by 1.0x	bilinear upsampling by 1.5x	bilinear upsampling by 2.0x
1.0x MNIST	**1.5x MNIST**	**2.0x MNIST**
structured conv, order 4, filters 16		
batch norm, relu		
2 x 2 max pool, stride 2	3 x 3 max pool, stride 3	4 x 4 max pool, stride 4
fully-connected, softmax		

MNIST a stride of 3 and a size of 3x3 is used and for the 2.0x MNIST a stride of 4 and a size of 4x4 is used. Table 5.1 shows an overview of architecture, including the dataset specific modifications that were made in the max pooling layer. Each architecture is trained with stochastic gradient descent (SGD) with a mini-batch of 100 samples, for 10 epochs.

The structured convolution layer has 16 structured respective fields (filters) that share the same sigma. By using multiple filters that share the same sigma, a single scale is learned that is stable to different feature initializations. It is stable because the sigma derivative is averaged over the alphas of all 16 filters. This allows us to repeat the experiment multiple times and compare the scales learned on the different datasets more easily.

RESULTS AND CONCLUSIONS

The main result of this experiment is shown in figure 5.6. It shows the development of the scale parameters during the training stage, when trained on the normal MNIST dataset and the resized versions: 1.5x MNIST and 2.0x MNIST. Each training run was repeated five times to show that the results are stable to different initializations. The average of these repetitions is plotted as a line, and colored region around it marks the range within one standard error.

These results show that feature scales are stable to different initialization and converge to a stable final value. However, the scales learned on the 1.5x MNIST and 2.0x MNIST datasets, are not exactly 1.5 and 2.0 times as large as the scale learned on the 1.0x MNIST dataset. This can be explained by the fact that sampling and resizing of continuous signal are not commutative operations. The scale-space theory describes the process of imaging a real physical space at a certain scale. The distance from the object to the imaging apparatus can vary continuously (rescaling), before the physical object is im-

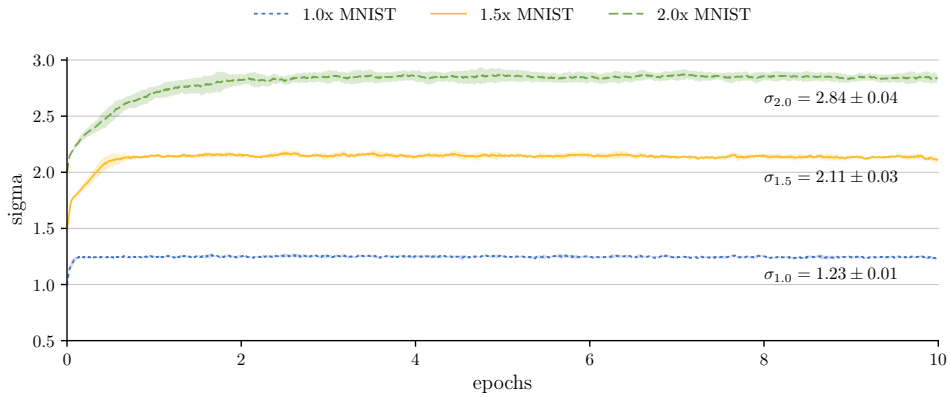


Figure 5.6: The development of the scale parameters during the training stage, when trained on the normal MNIST dataset and the resized versions 1.5x MNIST and 2.0x MNIST. The training runs are repeated five times to show that the results are stable to different initializations. The average of these repetitions is plotted as a line, and colored region around it marks the range within one standard deviation. The gaussian basis filters were created using the discrete derivatives. In figure 5.7 the same experiment is repeated, with the difference that the gaussian basis filters are created using the continuous derivatives.

aged on a discrete grid of pixels (sampling). But in this experiment, the original MNIST images were already sampled and then resized to form larger scale images. This causes the learned scale to differ slightly from what is expected according to scale-space theory.

This explains also why the ratio between the scale learned on the 1.5x MNIST and the 2.0x MNIST does correspond: $(2.0/1.5) \cdot \sigma_{1.5} = 2.81 \pm 0.04 \approx \sigma_{2.0} = 2.82 \pm 0.04$. These datasets both contain the same artifacts as a result of the resizing operation. This explanation can also be backed up by simulations in which a continuous gaussian is sampled, resized by a factor 1.5 and a factor 2.0, and then fitted again. The fitted sigma on the 2.0x resized gaussian is not exactly 2.0x the original sigma, but it is (2.0/1.5)x the sigma fitted on the 1.5x resized gaussian. This simulation is in agreement with the observations in this experiment. So, we conclude that this seems to be an artifact of how the images were created and that structured receptive fields are indeed able to learn the scale correctly during the training stage. The results of an experiment in the next chapter (section 6.1) also confirm this.

Section 4.8 mentions two different methods for creating the gaussian derivative filters: the discrete derivatives and continuous derivatives. Theoretically, the discrete derivatives should be preferred because the sampling of continuous expression for gaussian derivatives leads to discretization errors when sigma is small. For large sigmas, the continuous derivatives converge to the discrete derivatives and both methods produce the same basis filters. To show that this indeed the case, we repeated the experiment of figure 5.6 with the continuous derivatives. The results are shown in figure 5.7. We observe a slightly different value and a larger spreading for the scale learned on the MNIST 1.0x

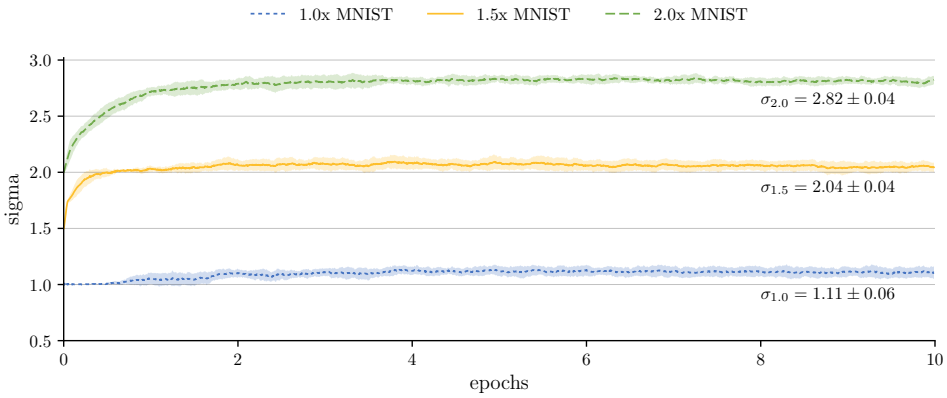


Figure 5.7: The development of the scale parameters during the training stage, when trained on the normal MNIST dataset (1.0x MNIST) and the resized versions 1.5x MNIST and 2.0x MNIST. The training runs are repeated five times to show that the results are stable to different initializations. The average of these repetitions is plotted as a line, and colored region around it marks the range within one standard error. The gaussian basis filters were created using continuous derivatives. In figure 5.6 the same experiment is repeated, with the difference that the gaussian basis filters are created using discrete derivatives.

(small sigma), while the value and the spreading for the scale learned on the MNIST 2.0x (large sigma) seems to be comparable to the discrete derivatives. This provides empirical evidence that is in perfect agreement with the hypothesis that discretization errors are large for small sigmas and negligible for large sigmas. We conclude that the discrete derivatives should indeed be preferred, especially when small sigmas are being considered, and we will use this method for the remainder of this report.

Finally, we demonstrate once more that normalization of the gaussian basis filters is of crucial importance. Figure 5.8, shows what happens when the experiment is repeated without normalizing the gaussian basis filters. The results clearly confirm that normalization is needed to learn the optimal features and their corresponding scales.

5.3. COMPARISON WITH NORMAL FILTERS

Now that our internal validation has been completed, it is time to investigate whether using structured receptive fields gives any advantage over using normal convolutional filters. To this end, we resemble the architecture that currently holds the state-of-the-art results on the CIFAR-10 and CIFAR-100 dataset, and ceteris paribus, we swap all the normal convolutional layers with structured convolution layers. The fact that the filter size of structured receptive fields is learned while the filter size of normal filters is fixed, should give structured receptive fields a small advantage. The purpose of this experiment, is to investigate how the classification performance of a state-of-the-art architecture is altered, when all its normal convolutional layers are replaced with structured convolutional layers.

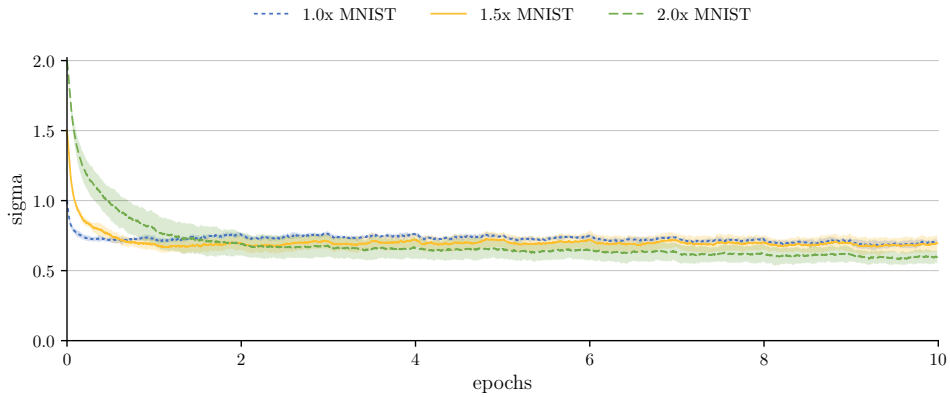


Figure 5.8: This is a repetition of the experiment in figure 5.6, without normalizing the gaussian basis filters. From this figure, it is clear that normalization of the gaussian derivatives (the basis filters) is absolutely crucial in order to learn the correct filters.

5

THE CIFAR DATASETS

The Canadian Institute For Advanced Research (CIFAR) has collected two labeled datasets of images [49] . These two CIFAR datasets consist of colored natural scene images, with 32×32 pixels each. The CIFAR-10 consists of images drawn from 10 classes and has 6000 images per class, while the CIFAR-100 images are drawn from 100 classes and consists of 600 images per class. Both datasets consist of 60,000 images and have been split into 50,000 training images and 10,000 test images while preserving an equal amount of images of each class in each set. The ten class labels in the CIFAR-10 dataset are: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck. The CIFAR-100 contains 20 similar classes, which are then split up into five finer classes. For example, it contains the superclass 'people', which has the following 5 subclasses: baby, boy, girl, man, woman. To preprocess the images, we subtract the channel means and divide by the channel standard deviations, both statistics are measured on the training set only.

DENSELY CONNECTED CONVOLUTIONAL NETWORKS

The architecture that currently holds the state-of-the-art results on the CIFAR datasets are the Densely Connected Convolutional Networks [47]. The key characteristic of DenseNets is the usage of so called 'dense blocks'. A dense block is a stack of normal convolution layers, in which each convolution layer takes as input the feature maps of all previous convolution layers in that block. In a dense block, each convolution layer is followed by a concatenation layer, which concatenates the input and output feature maps of the preceding convolution layer. This skip-connection gives the higher conv layers access to all previous feature maps in the dense block. This is different from traditional architectures, in which convolution layers only take as input the feature maps of the previous layer.

The original DenseNet architecture starts with a 3x3 convolution layer with 16 filters and

Table 5.2: Overview of the structured DenseNet architecture. The key characteristic of DenseNets is usage of a concatenation layer, which concatenates the input channels and output feature maps of the preceding conv layer. Differences with the original architecture: there are 6 instead of 12 conv layers per dense block and all normal conv layers (except the 1x1 conv in the transition blocks) have been replaced by structured conv layers. The dense blocks have been summarized to get the whole overview on a single page.

Layers	**Output size**	**Layer specific parameters**
CIFAR image	32 x 32 x 3	
structured conv	32 x 32 x 16	
Dense block 1 (6 layers)		
batchnorm, relu, structured conv 1, dropout	32 x 32 x 12	drop rate = 0.2
concatenate	32 x 32 x 28	
⋮	⋮	⋮
batchnorm, relu, structured conv 6, dropout	32 x 32 x 12	drop rate = 0.2
concatenate	32 x 32 x 88	
Transition block 1		
batchnorm, relu, 1x1 conv, dropout	32 x 32 x 88	drop rate = 0.2
average pooling	16 x 16 x 88	size 2x2, stride 2
Dense block 2 (6 layers)		
batchnorm, relu, structured conv 1, dropout	16 x 16 x 12	drop rate = 0.2
concatenate	16 x 16 x 100	
⋮	⋮	⋮
batchnorm, relu, structured conv 6, dropout	16 x 16 x 12	drop rate = 0.2
concatenate	16 x 16 x 160	
Transition block 2		
batchnorm, relu, 1x1 conv, dropout	16 x 16 x 160	drop rate = 0.2
average pooling	8 x 8 x 160	size 2x2, stride 2
Dense block 3 (6 layers)		
batchnorm, relu, structured conv 1, dropout	8 x 8 x 12	drop rate = 0.2
concatenate	8 x 8 x 172	
⋮	⋮	⋮
batchnorm, relu, structured conv 6, dropout	8 x 8 x 12	drop rate = 0.2
concatenate	8 x 8 x 232	
Final classifier		
batchnorm, relu, global average pool	232	
fully-connected, softmax	10 / 100	outputs class probabilities

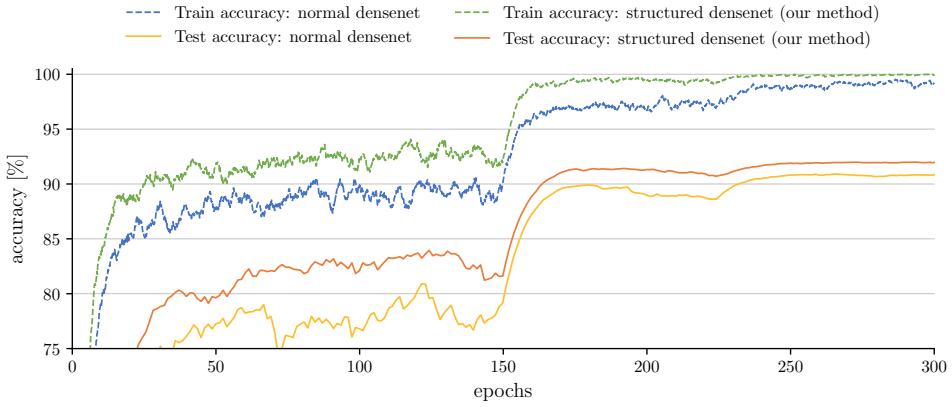


Figure 5.9: Development of the training and test set accuracy of the normal and the structured DenseNet on the CIFAR-10 dataset. The accuracy of the structured DenseNet is consistently higher than that of the normal DenseNet on both the training and the test set. This results in a final difference of ~1% in favor of the structured DenseNet.

is then followed by three dense blocks of twelve 3x3 convolution layers each. In between the dense blocks are so called 'transition blocks', in which the feature maps are subsampled to form images that are twice as small. A transition block consists of a 1x1 convolution layer followed by a 2x2 average pooling layer with a stride of 2. All convolution layers (except the first one) are preceded by a batchnorm layer and relu activation function, and are followed by dropout layer with a drop rate of 20%. The final block does the classification and consists of one last batchnorm layer and relu activation function, a global average pooling layer and a fully-connected layer. The softmax layer converts the outputs into proper class probabilities.

Because of hardware limitations, we had to reduce the number of conv layers in the dense blocks to six, before swapping all normal conv layers (except the 1x1 conv layers in the transition blocks) with structured conv layers. Table 5.2 summarizes of the resulting architecture, to which we will refer as 'Structured DenseNet'. We train both the Structured DenseNet and the Normal DenseNet (with 6 layers per dense block) on the CIFAR-10 dataset. To eliminate room for other explanations we use the same trainings procedure as the original paper [47]. We train with stochastic gradient descent using a mini-batch of 64 samples, for 300 epochs. The initial learning rate is set to 0.1, and is divided by 10 at 50% and 75% of the total number of training epochs. We use a Nesterov momentum of 0.9 without dampening, and a weight decay of 10^{-4} is applied to all parameters except the sigma of the structured conv layer.

RESULTS AND CONCLUSIONS

Figure 5.9, shows the main result of this experiment. It shows how the training and test accuracy of both architectures develop during the training on the CIFAR-10 dataset. We

Table 5.3: The first row shows the baseline result for the structured DenseNet in figure 5.9, and the corresponding experimental settings. In each of the subsequent rows one experimental setting has been changed with respect to the settings of the baseline result. For easy comparison also the baseline result and the relevant experimental setting is repeated in blue. From the test accuracies, it seems that the settings as used in figure 5.9 are already optimal. This does not come as a big surprise, since these settings were optimized for the original DenseNet architecture.

Test accuracy	Order	Alpha regularization	Sigma regularization	Dropout rate
92.04%	2	10^{-4}	0	0.2
89.94%	1			
92.04%	2			
91.52%	3			
90.44%		0		
92.04%		10^{-4}		
88.78%		10^{-3}		
92.04%			0	
91.94%			10^{-4}	
90.60%				0.0
92.04%				0.2

note that the accuracy of the structured DenseNet is consistently higher than that of the normal DenseNet on both the training and the test set. This results in a final difference of ~1% in favor of the structured DenseNet. Now it would not be wise to draw conclusions from on a single trainings run on one particular dataset, but this clearly demonstrates the practical potential of structured receptive fields. [2]

To eliminate room for other explanations we kept all experimental settings the same as the original DenseNet paper. However some of these settings might be highly tuned for normal convolution filters, and might not take into account the regularizing nature of structured receptive fields. Therefore, we investigate the effects on the classification performance of changing the some of these experimental settings. The following changes with respect to the baseline result in figure 5.9, are investigated: removing dropout; changing the regularization of the alpha parameters; adding regularization to the sigma parameters and changing the truncation order of the structured receptive fields. The final test accuracy and the corresponding experimental settings are shown in Table 5.3.

[2] A single trainings run takes about one week and requires a high-end server that contains multiple GPUs. We have only limited access to this hardware and renting this hardware in the cloud for such long periods is expensive. Hence, we were not able to investigate this potential as thoroughly as we would like to.

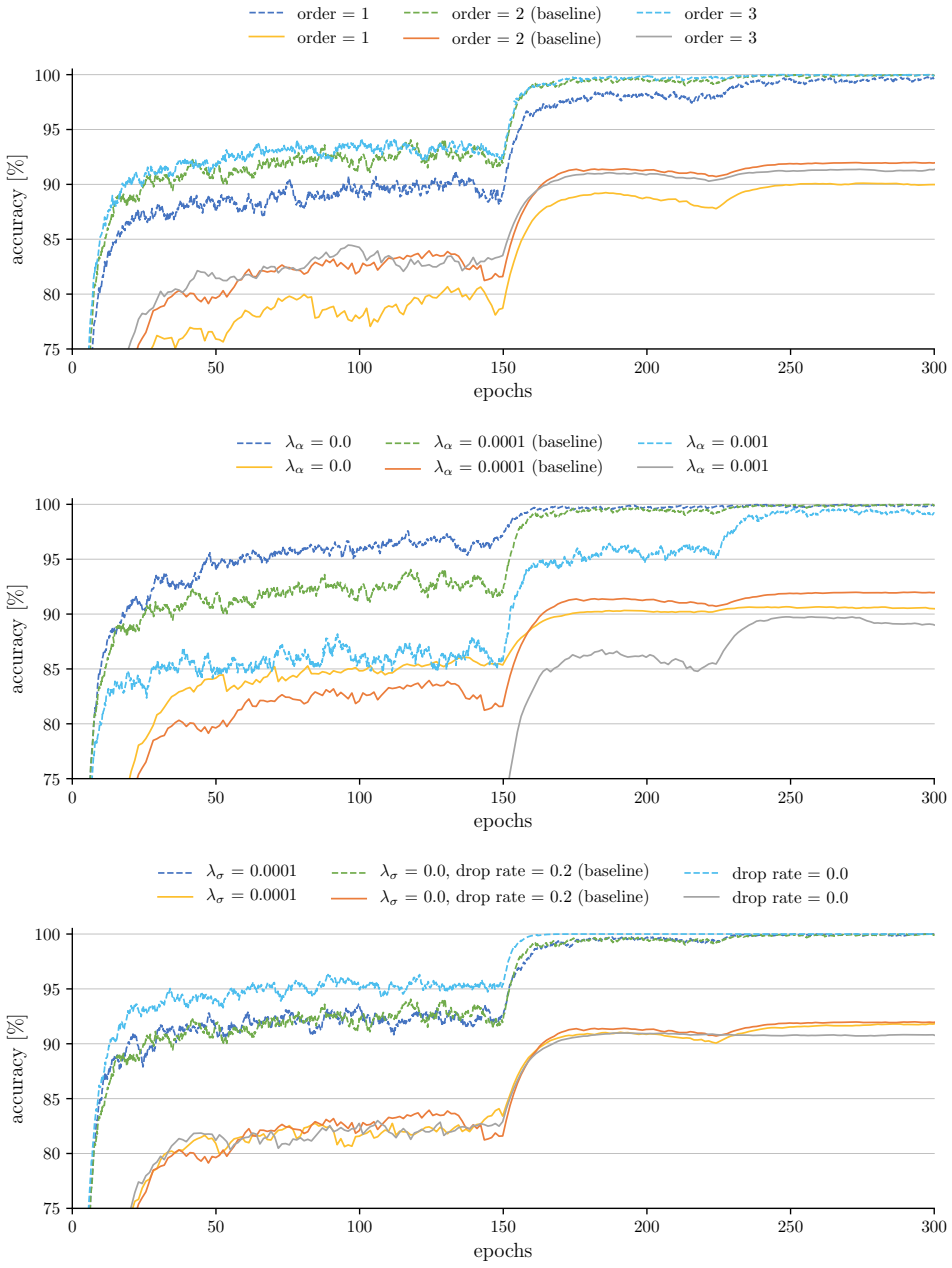


Figure 5.10: Training curves of a structured DenseNet on the CIFAR-10 dataset with different experimental settings. The top figure compares different truncation orders of the structured receptive fields; the middle figure compares different amounts of weight decay for the alpha parameters and the bottom figure shows the effect of removing dropout or adding weight regularization to the sigma parameters. The final test accuracy and the corresponding experimental settings are summarized in table 5.3. From these training curves, it seems that the baseline settings as used in figure 5.9, are already optimal.

From the test accuracies, it seems that the settings as used in figure 5.9 are already optimal. This does not come as a big surprise, since these settings were optimized for the original DenseNet architecture. However, we must stress that this search was limited to changing only one setting at a time. It could be that some settings dependent heavily on each other, and that changing them both at the same time will yield a significant improvement. However in view of time and the limited amount of computational resources, we have decided not to extend the sensitivity analysis beyond a 1-dimensional search.

For the sake of completeness, also the training curves corresponding to the results in table 5.3 are shown in figure 5.10. The top figure compares different truncation orders of the structured receptive fields; the middle figure compares different amounts of weight decay for the alpha parameters; the bottom figure shows the effect of removing dropout or adding weight regularization to the sigma parameters.

6

EXTRA APPLICATIONS

There are no such things as applied sciences, only applications of science.

Louis Pasteur

THE previous chapter demonstrated that structured receptive fields are more or less normal filters which can learn their own filter size and that this extra ability over normal filters seems to give them an advantage when used in classification tasks. This advantage over normal filters is in itself a valid application of structured receptive fields.

In this chapter however, we would like to show two extra use-cases of structured receptive fields. These applications arise from the fact that structured receptive fields, in contrast to normal CNN filters, have an explicit and continuous scale parameter that is learned during the training stage. Hence, these applications are unique to structured receptive fields and cannot be realized (easily) by normal filters.

Section 6.1 shows that structured receptive fields can be used to build an architecture that is able to classify and localize/detect objects over a range of different scales, while being trained on a single scale only. In other words, the global scale at which the CNN operates can be changed by adjusting the scale parameters of all the structured receptive fields.

Section 6.2 shows that the learned scale parameters of structured receptive fields can be used to investigate the role/purpose of subsampling layers. Comparing the classification accuracies and the learned scale parameters of an architecture that is trained both with and without subsampling layers, reveals new insights about the role and workings of the subsampling layer, which has been so poorly understood until now.

The purpose of this chapter is not to give a detailed and thorough investigation of these two use-cases. Instead, we will explore these applications to encourage and to give guidance to further research on structured receptive fields. We believe that both these applications could easily span an entire research, if one wants to investigate these topics thoroughly.

6.1. SELECTING THE GLOBAL OPERATION SCALE OF CNNS

Normal CNNs used for image classification can only handle input images of a single size. The input image is first resized to the predefined size, before the CNN takes it as input and classifies it. Normal CNN filters are therefore trained to operate at a single predefined scale which, in contrast to structured receptive fields, cannot be changed explicitly.

In this experiment, an architecture is built that will be trained on images of a single predefined scale, just like a normal CNN. However, it can then used to evaluate images at different scales. This is possible because we can scale the learned sigma parameters of all the structured receptive fields in the architecture. In other words, the global scale at which the CNN operates can be changed by scaling all the learned sigma parameters by the same factor. So, instead of resizing the input image to the scale of the CNN, the CNN is rescaled to the size of the image.

NETWORK ARCHITECTURE AND TRAINING

A larger input image, does not necessarily mean that the object in the image appears at a larger scale. The object could for example not be covering the entire image. Therefore, the proposed architecture should be able to classify the same object regardless of any scaling and translation. An architecture that can do this, is said to invariant to these transformations: the output (the class label) is the same, regardless of whether any scaling or translation is applied to the input image. Note that classification is by definition invariant: it reduces input signals to the same label regardless of intra-class variability.

Although the CNN as a whole must be invariant, we argue that the feature extraction part should be equivariant with respect to scalings and translations, because it is impossible to determine whether features are in the right spatial configuration if they are invariant. We argue that the invariance should be introduced no earlier than the classification layers at the very end of the CNN.

Before continuing to explain the used architecture, it is important to understand the difference between equivariance and invariance. A function is equivariant to a group of transformations, if interchanging the order of transformation and applying the function gives the same output, for every possible input and transformation of the group. A function is said to be invariant to a group of transformations, if the output of the function is the same regardless of any transformation of the group being applied to the input.

Table 6.1: Overview of the CNN architecture used in the experiment in section 6.1

Layers	Output size (training)	Output size (testing)	Layer specific parameters
MNIST image	28 x 28	? x ?	
structured conv	28 x 28 x 16	? x ? x 16	order 4, filters 16
batchnorm, relu	28 x 28 x 16	? x ? x 16	
structured conv	28 x 28 x 32	? x ? x 32	order 4, filters 32
batchnorm, relu	28 x 28 x 32	? x ? x 32	
structured conv	28 x 28 x 10	? x ? x 10	order 4, filters 10
batchnorm, relu	28 x 28 x 10	? x ? x 10	
global max pool	10	10	
softmax	10	10	outputs the class probabilities

A total equivariant feature extraction requires intermediate layers that are also equivariant. Hence, we propose an architecture which is solely composed of layers that are equivariant with respect to scaling and translation of the input image. A normal convolutional layer is already equivariant to translation, but not to scaling. In order to make it equivariant to scaling, the filters should be scaled. We note that structured convolution layers can do this by scaling the sigma with same factor as the input image. Furthermore, pointwise operations such as relu activations and batch normalization are already equivariant [58], but pooling layers are not!

In this experiment, a feature extraction is used that consists of three blocks of structured conv layers with batch normalization and relu activation functions, but without pooling layers in between. A global max pool layer is used to make the resulting feature representation invariant for classification. It reduces all final feature maps to a single value: the maximum intensity of the feature map. The fact that this reduction works regardless of the size of the feature map, enables the CNN to handle input images of any size. A final softmax layer is used convert all feature map maxima into proper class probabilities. A complete overview of the used architecture is shown in table 6.1.

The network is trained on the original MNIST dataset with stochastic gradient descent with a mini-batch of 100 samples, for 20 epochs.

The reason for not using fully-connected layers, is that the final feature maps are not exactly equivariant to the different transformations. Discretization errors because of the continuous transformations on a discrete grid, cause the feature maps to be slightly different for different transformations. We already saw in the experiment in section 5.2, that these discretization errors caused our hypothesis for the values of the learned scales to deviate from the actual values.

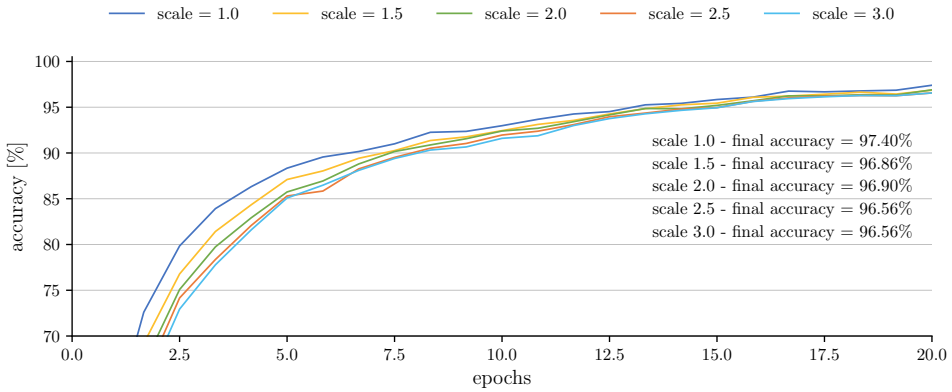


Figure 6.1: The development of the test accuracy, during the training stage for different scaled versions of the MNIST testset. From these training curves it is clear that the proposed architecture in table 6.1 can classify the same objects over a wide range of different scales by scaling the sigma parameters of all the structured receptive fields in the architecture.

The maximum intensity of the final feature maps is therefore not stable under different transformations and using fully-connected layers on top of this unstable invariant would cause it destabilize even further. Fortunately, the argmax over all feature map maxima is a more stable invariant. In other words, the maximum intensity of each feature map varies for different transformations, but the highest maximum keeps occurring in the same feature map. Therefore we let each final feature map correspond to a particular class, and use the argmax of the feature map maxima to predict the right class. A final softmax layer is used to convert all feature map maxima into proper class probabilities.

RESULTS AND CONCLUSIONS

The trained architecture is evaluated on the original images of the MNIST testset and on resized versions. Before evaluation, the learned sigma of every structured convolutional layer is multiplied with the same resize factor as the images. This global scale selection ensures that the structured receptive fields operate at the correct scale. Figure 6.1 shows the development of the test accuracy, during the training stage for different scaled versions of the MNIST testset.

The final accuracy of 97.40% on the original MNIST testset, does not come close to the state-of-the-art on the MNIST dataset which is 99.77% [59], but that was never the purpose of this experiment. Though, we are confident that one could easily improve the test accuracy by either making the architecture deeper (it now only contains 12,000 parameters) or by training the network to full convergence (the test accuracy still improved significantly over the last 5 epochs). The purpose of this experiment is to show the architectures ability to do scale and translation invariant image classification. From the training curves in figure 6.1, it is clear that the proposed architecture is able to do this if the right global scale is selected.

	Scale = 1.0	Scale = 1.5	Scale = 2.0	Scale = 2.5	Scale = 3.0
y = 0	0.19%	0.01%	0.00%	0.00%	0.00%
y = 1	0.83%	0.11%	0.02%	0.01%	0.00%
y = 2	0.19%	0.03%	0.01%	0.00%	0.00%
y = 3	0.06%	0.00%	0.00%	0.00%	0.00%
y = 4	**56.35%**	49.25%	49.40%	41.94%	47.24%
y = 5	0.06%	0.00%	0.00%	0.00%	0.00%
y = 6	1.44%	0.13%	0.03%	0.01%	0.00%
y = 7	0.12%	0.00%	0.00%	0.00%	0.00%
y = 8	0.40%	0.02%	0.00%	0.00%	0.00%
y = 9	40.35%	**50.46%**	**50.54%**	**58.05%**	**52.76%**

Figure 6.2: The same handwritten digit at 5 different scales: 1.0, 1.5, 2.0, 2.5 and 3.0. Below each image are the predicted class probabilities. The highest probability is the predicted class and is highlighted in **bold**. The digit is correctly classified at the original scale (first column), but misclassified as a '9' at larger scales.

The final test accuracy for the larger scale is ~1% lower than the test accuracy on the original scale. This probably because the used invariant is not stable enough to the scaling transformation, for all the examples in the testset. Figure 6.2 shows such an example, with the predicted class probabilities at different scales. The class probability of the correct label is already quite close to the misclassified label at the original scale. This is possibly because the network was not trained to full convergence. The instability due to discretization errors in the scaling transformation, then causes the label to change from the correct class to the wrong class at larger scales.

We conclude that the loss in performance is small and might decrease if one would train the network architecture to full convergence. Furthermore, we note that the performance loss was much larger when we used fully-connected layers on top of the global max pooling layer. We suspect that the instability due to discretization errors of the scaling, is blown up by the fully-connected layers. In general, we conclude that this architecture is able to classify the same object over a range different scales, if the right global scale is selected. Although the scale selection is not done automatically, providing CNNs with the ability to select the scale at which they operate is an important step towards scale invariant image classification.

In future research, one could investigate the possibility of selecting the global scale automatically, instead of a manually. For example, by using a CNN that is specifically trained to select the right scale or by using the automatic scale selection procedure described by scale-space theory. Scale-space theory covers the automatic scale selection for image features that are described in terms of gaussian derivatives. This should be directly applicable to structured receptive fields which are linear combinations of gaussian derivatives. Furthermore, the automatic scale selection could be implemented in an efficient manner, since selecting the scale of the first layer is enough to select the scale of the whole CNN: the ratio between the scales of the different layers is fixed and learned during the training stage.

LOCALIZATION AND DETECTION

Finally, we explore whether the final feature map could be used as a building block for localization and detection tasks. The location of the maximum in the final feature map should roughly correspond to the location of the object, while the global scale factor should correspond roughly to the extend of the object. The omission of intermediate pooling layers ensures that the location of the object is immediately reflected in the location of the maximum (without needing any upscaling). Figure 6.3 shows some MNIST examples which are scaled, translated and distorted with 80% salt-and-pepper noise. The final feature map corresponding to the predicted class is shown next to each example. The center of the circle corresponds to location of the maximum, and the radius corresponds to the global scale factor. From these examples, it seems that the final feature maps look promising as building block for a new approach to localize and detect objects. However much more research is needed, to draw hard conclusions.

In future research, one could investigate whether this method also works on more challenging datasets than MNIST.

6.2. INVESTIGATING SUBSAMPLING

Neurons in a normal CNN have an increasingly larger effective receptive field as depth progresses. In other words, the activation of a particular neuron is a function of a larger and larger patch of pixels in the input image, as the neuron is situated in a deeper layer of the CNN. For example, the activation of a neuron in a 3x3 convolutional layer is a function of a 3x3 local receptive field in the input image. If another 3x3 convolutional layer is stacked on top of this layer, the neurons of that layer would have an effective receptive field of 5x5 in the input image. One would need approximately 16 3x3 layers to have an effective receptive field that covers an entire 32x32 image or 128 3x3 layers to cover an entire 256x256 image. This obviously means that one needs a lot of compute power to compute all these layers, and that one needs large datasets to learn all the free parameters associated with those layers.

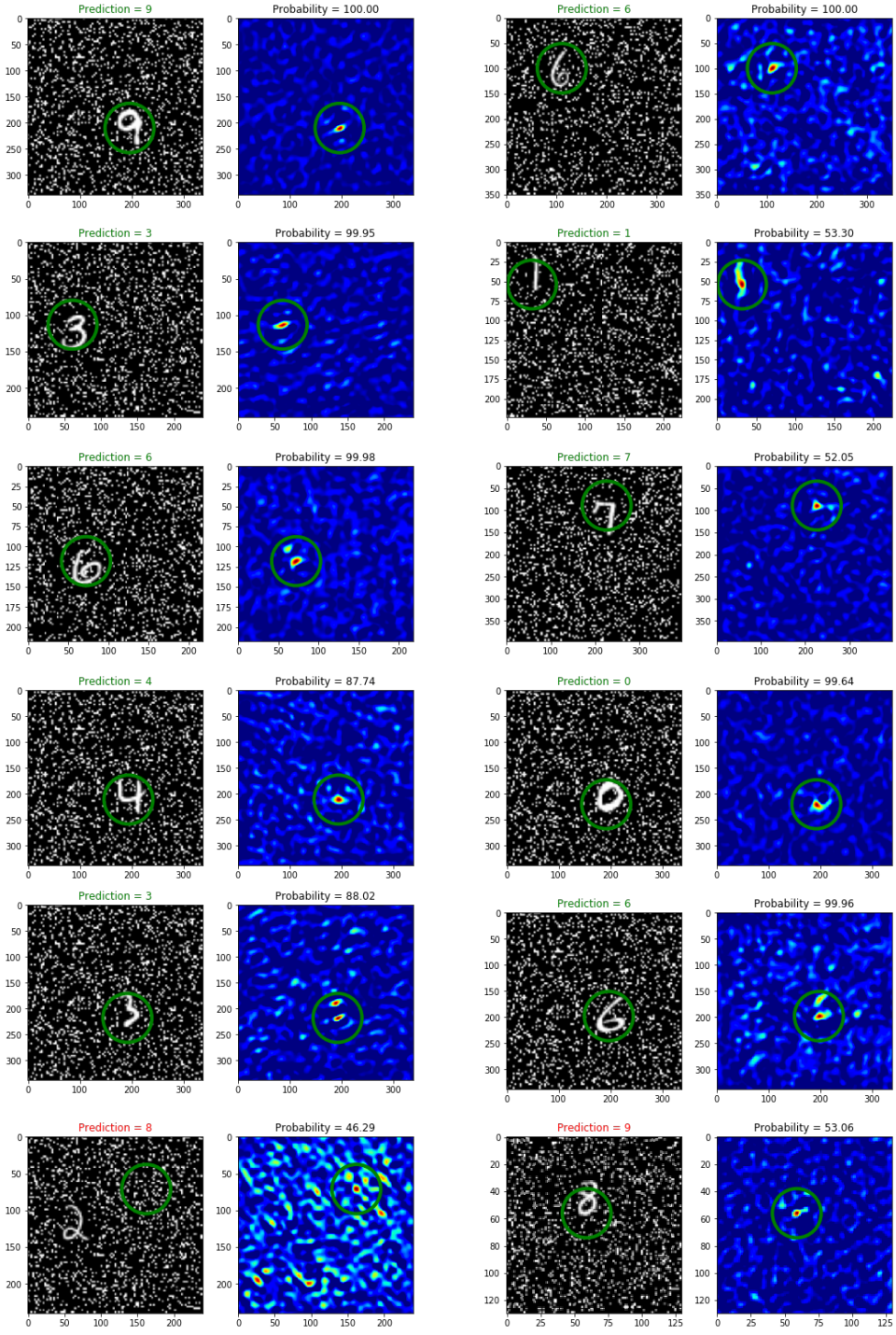


Figure 6.3: 12 examples of MNIST digits which are scaled, translated and distorted with salt-and-pepper noise. A significant drop in the classification accuracy was observed after adding 80% salt-and-pepper noise to the image. The bottom row shows two examples that were classified correctly before adding the salt-and-pepper noise, but are misclassified after the noise was added.

One way to reduce the number of needed layers and therefore also the number of learnable parameters, is to use subsampling layers. Historically the most commonly used subsampling layers are average and the max pooling. However, since *Springenberg, et al.* [52] showed that convolutional layers with a stride of two work just as good, there has been a trend to use this instead. Which exact layer is used, does not matter too much since the effect is the same. They all reduce the spatial size of the feature maps with a factor of two and thereby increase the effective receptive field of the subsequent convolutional neurons in an exponential fashion. E.g. stacking N consecutive subsampling layers reduces the spatial size of the feature maps with a factor 2^{-N} and thereby increases the effective receptive field with a factor 2^N.

The usage of subsampling layers allows a rapid (exponential) growth of the effective receptive fields of the subsequent convolutional layers, without needing to increase the number of learnable parameters. Structured receptive fields can be used to investigate whether this is indeed the main function of the subsampling layers. This is possible, because the filter size of structured receptive fields is learned during the training stage and because their filter size can be increased without increasing the number of learnable parameters.

This will be put to the test, by training an architecture containing structured convolutional layers, once with subsampling layers and once without subsampling layers. If the sole purpose of the subsampling layers is to increase the receptive field size of the subsequent convolutional layers without increasing the number of learnable parameters, then the structured receptive fields should be able to compensate for the omission of the subsampling layers by learning larger sigmas and therefore larger effective receptive fields. In other words, the test accuracy of both architectures should be the same, because the structured receptive fields can take over the role of the subsampling layers.

As a sanity check, the experiment is repeated with normal convolutional layers. Because they cannot take over the role of the subsampling layers by adopting their filter size, we expect a worse test accuracy in the architecture in which the subsampling has been omitted.

THE NETWORK-IN-NETWORK ARCHITECTURE

For this experiment, the *Network-in-Network* (NIN) architecture will be used [60]. This is mostly because this architecture is small enough to fit in the GPU memory, even when subsampling is omitted. Note also that, a Densenet or Resnet would not be suitable for such an experiment because of the complicated ways in which the effective receptive fields depend on the skip-connections in the architecture.

The original NIN architecture contains three blocks, that the authors call: 'multi layer perceptron convolution blocks' (MLPconv). Each MLPconv block consists of one spatial convolutional layer, followed by two non-spatial 1x1 convolutional layers. According to the authors, these blocks resemble a two-layer fully-connected neural network in a con-

Table 6.2: An overview of the four different versions of the Network in Network architecture in one table. One can obtain any of the four architectures by making two decisions: the architecture either uses normal or structured convolutional layers as first layer of a block and the architecture either uses a 3x3 max pooling layer in the transition blocks to subsample the image, or it does not. Each architecture is trained on the CIFAR-10 dataset and the development of the train and test accuracy are shown in figure 6.4. The bias terms and relu activations after every normal or structured convolutional layer, are not shown in this table.

Layers	**Output size (with / without subsampling)**
CIFAR-10 image	32 x 32 x 3
MLPconv block 1	
5x5 conv / structured conv, order 2	32 x 32 x 192
1x1 conv	32 x 32 x 160
1x1 conv	32 x 32 x 96
Transition block 1	
(3x3 max pooling, stride 2	16 x 16 x 96)
dropout, drop rate = 0.5	16 x 16 x 96 / 32 x 32 x 96
MLPconv block 2	
5x5 conv / structured conv, order 2	16 x 16 x 192 / 32 x 32 x 192
1x1 conv	16 x 16 x 192 / 32 x 32 x 192
1x1 conv	16 x 16 x 192 / 32 x 32 x 192
Transition block 2	
(3x3 max pooling, stride 2	8 x 8 x 192)
dropout, drop rate = 0.5	8 x 8 x 192 / 32 x 32 x 192
MLPconv block 3	
3x3 conv / structured conv, order 2	8 x 8 x 192 / 32 x 32 x 192
1x1 conv	8 x 8 x 192 / 32 x 32 x 192
1x1 conv	8 x 8 x 10 / 32 x 32 x 10
Final classifier	
global average pool, softmax	10

volutional neural network, hence the name Network-in-Network. The filter sizes of the spatial convolutional layers in these three blocks are respectively 5x5, 5x5 and 3x3, and were optimized by the authors using a validation set. All convolutional layers are followed by a bias term and a relu activation function. In between the MLPconv blocks are transitions blocks in which the image is subsampled using a 3x3 max pooling layer with a stride of 2. Also a dropout layer with a droprate of 50% is added to improve generalization. There are 10 final feature maps that all correspond to a particular class. A global average pooling layer determines the average of each feature map, and the feature map with the heighest average is the predicted class. A final softmax layer is used convert all averages into proper class probabilities.

For this experiment we modify the original NIN architecture by swapping the spatial convolutional layers in the MLPconv blocks for structured convolutional layers of order 2. The resulting architecture is trained once with max pooling layers, and once without max pooling layers. As a sanity check, we also train the original architecture, once with and once without max pooling layers . The four used architectures are summarized in table 6.2.

The same training procedure is used as the original NIN paper. The architectures are trained on the CIFAR-10 dataset using gradient descent for 300 epochs, with a mini-batch of 128 samples. The CIFAR-10 images are normalized by the channel means and standard deviations of the training set. The initial learning rate is set to 0.001, and is divided by a factor 10 after 150 training epochs. We use the Adam update rule with a β_1 of 0.9 and a β_2 of 0.999, and a weight decay of 10^{-4} is applied to all parameters except the sigmas of the structured conv layer and the bias variables.

RESULTS AND CONCLUSIONS

Figure 6.4 shows the development of the train accuracy and test accuracy for the different architectures. The different architectures are plotted with different colors (see legend) and the train and test accuracy are indicated by a dashed and a solid line, respectively. From this plot it is clear that the effect of removing the subsampling layers, is a far greater for the architecture that uses normal convolutional layers than for the architecture that uses structured convolutional layers. Indicating that the main purpose of the subsampling layers is indeed increasing the effective receptive fields of the subsequent convolutional layers.

The way in which the sigma parameters of the structured convolutional layers develop, seems to support this claim. Figure 6.5 shows the development of the sigma parameters of the three structured convolutional layers, for both the architecture with and without pooling layers. The value of the sigma parameter of the first convolutional layer (before the first pooling layer) is approximately the same for both architectures, but for the other two structured convolutional layers the values are much larger in the architecture where the pooling layers have been omitted. This is in line with our hypothesis that structured receptive fields can, at least partially, take over the role of subsampling layers by learning

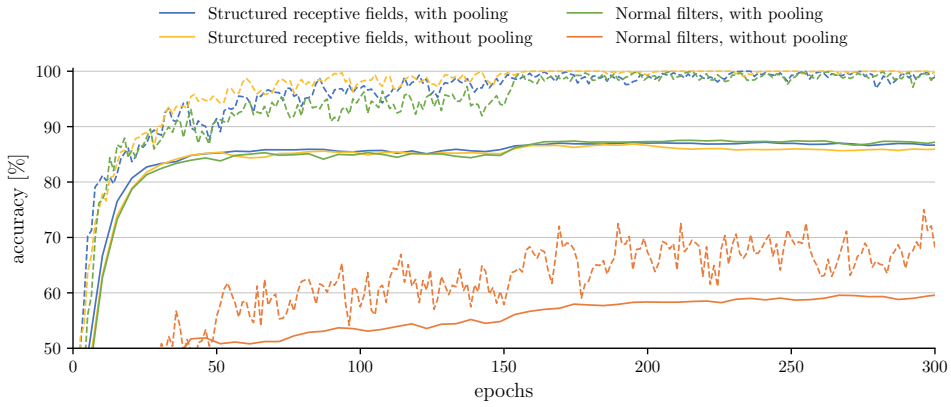


Figure 6.4: Development of the train accuracy and test accuracy for the different architectures. The different architectures are plotted with different colors (see legend) and the train and test accuracy are indicated by a dashed and solid line, respectively. From this plot it is clear that the effect of removing the subsampling layers is a far greater for the architecture that uses normal convolutional layers, than for the architecture that uses structured convolutional layers.

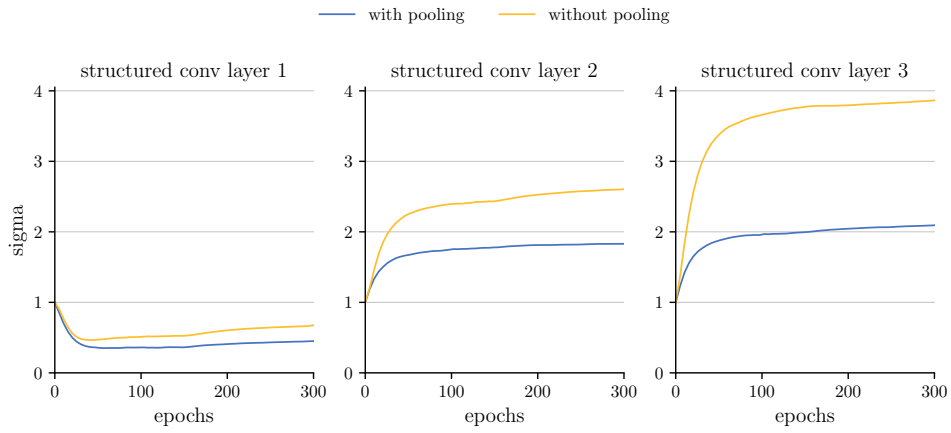


Figure 6.5: A comparison of the sigma parameters in the structured convolutional layers, between the architecture with pooling layers and the architecture without pooling layers. The value of the sigma parameter of the first convolutional layer (before the first pooling layer) is approximately the same for both architecture, but for the other two structured convolutional layers the values are much larger in the architecture where the pooling layers have been omitted. This is in line with our hypothesis that structured receptive fields can, at least partially, take over the role of subsampling layers by learning larger sigmas and therefore larger effective receptive fields.

larger sigmas and therefore larger effective receptive fields.

From these results it is clear that the main purpose of the subsampling layers is to increase the effective receptive fields of the subsequent convolutional layers. However, this is most likely not the only function of subsampling layers because, although smaller than for normal filters, the test accuracy also drops for structured receptive fields after removing the subsampling layers.

In the process of increasing the effective receptive fields by subsampling the image, the subsampling layers also throw away information and lower the dimensionality of the feature representation. Our guess is that this has a regularizing effect and therefore improves generalization. This hypothesis is supported by the observation that the structured architecture without pooling layers achieves a higher training accuracy more easily than the architecture with pooling layers, but that the test accuracy for the architecture without pooling layers is significantly lower. See the blue and yellow lines in figure 6.4. This indicates that the architecture without pooling layers overfits more easily on the training set than the architecture with pooling layers.

This could be investigated by training the architecture once more. This time removing the subsampling layers, and adding an operation which sets all even rows and even columns to zero to get a similar effect as throwing away information by subsampling. If the test accuracy becomes approximately the same as the architecture with pooling layers, then we can conclude that subsampling layers have two important functions: increasing the effective receptive field of all subsequent convolutional layers (without increasing the number of learnable parameters) and acting as a structural regularizer to improve generalization by throwing away information (subsampling effectively sets ~75% of the features to zero).

Unfortunately, there is no time left to investigate this any further. Nevertheless, we feel that the two applications presented in this chapter show valid and interesting applications of structured receptive fields. The experiments in this chapter, serve the purpose of giving guidance for further research. We sincerely hope that after seeing these applications, other researchers also see the potential of structured receptive fields and will continue our work.

IV

Wrap-up

7

CONCLUSIONS

The conclusion is the point where you get tired of thinking.

Arthur Bloch

THIS chapter summarizes the results of this research and thereby tries to answer the research questions as they were stated in the introduction of this report. The research questions are repeated below to refresh the readers memory.

Main question

Can structured receptive fields replace normal filters in any CNN architecture with an always strictly better or equal performance in classification tasks?

Proof of concept

1. Can structured receptive fields approximate all the filters that learned by normal CNNs, despite their imposed mathematical form?
2. It is possible to learn the feature scale (filter size) of structured receptive fields during the training stage?
3. Does the usage of structured receptive fields provide an advantage over using normal filters, in classification tasks?

Extra applications

1. Can the learned scale parameters be adjusted manually after the training, to change the global scale at which the whole CNN operates? I.e. is the CNN able to classify the same image correctly over a range of different scales while being trained on images of a single scale only, if we select the right global scale?
2. Is it possible to use the learned scale to investigate the role of subsampling layers in CNNs? I.e. when subsampling layers are omitted, do the structured receptive fields learn larger feature scales for the subsequent convolutional layers and what happens to classification accuracy?

7.1. PROOF OF CONCEPT

The proof of concept in chapter 5, consists of three experiments that together aim to demonstrate the practical potential of structured receptive fields. This section summarizes the main results and conclusions of the proof of concept.

1. Representational power. The results in section 5.1, show that structured receptive fields can approximate the filters of a pre-trained AlexNet architecture. Therefore it seems plausible that they are also able to approximate the filters learned by other CNN architectures. In the current implementation, it is needed that structured receptive fields in the same convolutional layer share the same truncation order and sigma. Fortunately, the quality of these approximations does not drop by a large amount when these restrictions are enforced. The effect on the classification performance however, can be investigated further. Though this will require some serious computational power.

Furthermore we observe: that a truncation order of 4 is sufficient to approximate most of the AlexNet filters; that the weights of the structured receptive fields range from -0.1 to 0.1 and that normalization of the gaussian derivatives is crucial for gradient descent to converge to the correct approximations. This is useful knowledge that can be used in following experiments. Finally, we note also that due to their mathematical structure, structured receptive fields require less parameters than normal filters to represent the same effective filter. This is especially true for large filters and seems to be in agreement with the conclusion of *Jacobsen et. al* [32] that structured receptive fields learn better than normal filters, when the amount of training data is limited.

2. Learning the feature scale. The results in section 5.2, show that structured receptive fields can indeed learn a meaningful feature scale. The scale learned on the normal and resized images of the MNIST dataset, converges to a final value that is stable to different initializations. The fact that the scale learned on the 1.5x and 2.0x enlarged images is not 'exactly' 1.5x and 2.0x as large as the scale learned on the original images, is explained as an artifact of the way in which the images were created. This artifact results from the fact that sampling and resizing of continuous signal are not commutative operations. This explanation is also supported by the results of a simple simulation in which a continuous gaussian is sampled, resized, and fitted again. The results of the simulation are in agreement with the observations in the experiment. Therefore, we conclude that structured receptive fields are indeed able to learn a meaningful feature scale.

Finally, we repeated the experiment once using a different method to create the gaussian derivative filters, and once using the same method but without normalizing the gaussian derivative filters. From these results it is obvious that normalization of the gaussian derivatives is crucial to be able to learn a meaningful scale. Furthermore we see that the discrete gaussian derivatives are to be preferred over the continuous derivatives. The absence of discretization errors in the discrete derivatives, lead to feature scales that are numerically more stable, especially when sigma is small. When sigma is large, the discretization errors in the continuous derivatives are negligible and both methods produce approximately the same effective filters. Therefore both methods also learn approxi-

mately the same feature scales.

3. Comparison with normal filters. The result in section 5.3, shows that using structured receptive fields seems to give an advantage over using normal filters. By replacing normal filters in a DenseNet architecture and keeping everything else the same, a ~1% higher test accuracy was obtained on the highly competitive CIFAR10 benchmark dataset. It would not be wise to draw conclusions from a single trainings run on one particular dataset, but these results definitely show practical potential of structured receptive fields and this is after all, the purpose of this proof of concept.

Finally, the performed sensitivity analysis of the experimental settings shows that the experimental settings that were used for the original densenet architecture, are also optimal for the structured densenet.

Final word. Let us now summarize all of the above in one single sentence. The three experiments in this proof of concept show that structured receptive fields are more or less normal filters (section 5.1) which can learn their own filter size (section 5.2) and that this extra ability over normal filters seems to give them an advantage when used in classification tasks (section 5.3).

7.2. EXTRA APPLICATIONS

The applications in chapter 6 demonstrate two extra use-cases of structured receptive fields, on top of the potential advantage that has already been demonstrated in the proof of concept. These extra use-cases arise from the fact that structured receptive fields, in contrast to normal CNN filters, contain an explicit scale parameter that is learned during the training stage. Hence, these applications are unique to structured receptive fields and cannot be realized (easily) by normal filters.

The purpose of chapter 6 is not to give a detailed and thorough investigation of the two use-cases. Instead, the applications serve to encourage and to give guidance for future research on structured receptive fields.

1. Selecting the global operation scale of CNNs. The results in section 6.1, show that structured receptive fields can be used to build an architecture of which the scale of operation can be changed, after the training stage. This allows the training of an architecture on images at one scale, and then 'scale up' the CNN to evaluate images on an other scale. The results show that, if the right global scale is selected, the proposed architecture is able to classify examples of the MNIST test set correctly over a wide range of different scales (1.0x, 1.5x, 2.0x, 2.5, 3.0x), while being trained only on the original training images (scale 1.0x). Like the second experiment of the proof of concept, this experiment also confirms the fact that structured receptive fields can learn a meaningful feature scale during the training stage.

There is a small drop in classification accuracy (< 1%) for the test sets at larger scales, but

just like the experiment in section 5.2, this seems to be an artifact of the way in which images are resized. Furthermore, it could well be that this drop in performance diminishes as the network is trained to full convergence.

Additionally, we took a brief look at the final feature maps to see whether they could be used as a building block for localization and detection tasks, since their signal is equivariant to scaling and translation.. The location of the maximum in the final feature seems to correspond roughly to the location of the object, while the global scale factor corresponds to the size of the object. The omission of intermediate pooling layers, ensures that the object location is reflected immediately (without the need to upscale) in the location of the maximum in the final feature map. From the examples we looked at, it seems that the final feature maps look promising as building block for a new approach to localize and detect objects. However more research is needed, to be able to draw hard conclusions.

All in all, we conclude that these types of architectures with structured convolutional layers and without pooling layers, look promising for both translation and scale invariant classification, as well as localization and detection tasks. In future research, one could investigate the possibility of selecting the global scale automatically, instead of manually.

2. Investigating subsampling. The results in section 6.2, show that the learned scale parameter can be used to investigate the role and importance of subsampling in CNNs. Two versions of the Network-in-Network architecture: one with normal convolutional layers and one with structured convolutional layers, are both trained with, and without subsampling layers. From the drop in the test accuracy it is clear that the effect of removing the subsampling layers is far greater for the architecture that uses normal convolutional layers (~28%) than for the architecture that uses structured convolutional layers (~2%). Indicating that the main purpose of the subsampling layers is indeed increasing the effective receptive fields of the neurons in the subsequent layers.

The way in which the sigma parameters of the structured convolutional layers develop, seems to support this claim. The value of the sigma parameter of the first convolutional layer (before the first pooling layer) is approximately the same for both the architecture with and without subsampling layers, but for the other two structured convolutional layers the values are roughly 1.5-2.0 times as large in the architecture where the subsampling layers have been omitted. This is in line with our hypothesis that structured receptive fields can, at least partially, take over the role of subsampling layers by learning larger sigmas and therefore larger effective receptive fields.

Increasing the effective receptive field is may not be the only function of subsampling layers, because, although smaller than for normal filters (~28%) the test accuracy also drops with ~2% for structured receptive fields after removing the subsampling layers. In the process of increasing the effective receptive fields by subsampling the image, the subsampling layers also throw away information and lower the dimensionality of the feature representation. Our guess is that this might have a regularizing effect and there-

fore improves generalization.

In future research, this could be investigated by training the architecture without subsampling layers once more and adding an operation which sets all even rows and even columns to zero to get a similar effect as throwing away information by subsampling. If the test accuracy becomes approximately the same as the architecture with pooling layers, then we can conclude that subsampling layers have two important functions: increasing the effective receptive field of all subsequent convolutional layers and acting as a structural regularizer to improve generalization by throwing away information.

Final word. We feel that the two applications presented in chapter 6 show useful and interesting applications of structured receptive fields that cannot be realized with normal filters. The experiments serve the purpose of giving guidance for future research and we sincerely hope that after seeing these applications, other researchers also see the potential of structured receptive fields and will continue on our work.

7.3. MAIN RESEARCH QUESTION

Now that all subquestions have been answered, the time has come to answer the main question of this research: *Does replacing normal filters with structured receptive fields always lead to better or equal performance?*

Unfortunately the results in this report are not sufficient, to answer this question conclusively. For a DenseNet architecture on the CIFAR10 dataset replacing normal filters with structured receptive fields results in a ~1% better performance, but this does not guarantee anything for other architectures or other datasets. In order to answer the main question conclusively, this experiment needs to be repeated on many different architectures and datasets [1].

Although we cannot not predict the outcome of these experiments, the proof of concept experiments indicate that structured receptive fields are more or less normal filters with the extra ability to learn their own scale (filter size). Hence, it can be expected that this extra ability will improve the performance in cases where wrong filter sizes have been selected, while the performance might stay the same for cases in which the filter sizes are optimal already.

However, until this has been investigated it remains nothing more than a hypothesis or an educated guess.

[1] There was simply not enough time nor hardware to do this during the course of this research. So, instead we focussed on laying down a foundation that brings us a step closer to answering the main question, and which provides a clear roadmap for answering the main question in the (near) future.

8

DISCUSSION AND RECOMMENDATIONS

The important thing is not to stop questioning.
Curiosity has its own reason for existing.

Albert Einstein

THE previous chapter summarized the results and conclusions of this report. In doing so, it answered the sub and main questions of this research. This chapter on the other hand, discusses the importance and consequences of these answers for the scientific world. It discusses the contribution of our work, the limitations of our method, the things done differently in hindsight and it recommends a few promising and interesting ways to continue this research in future.

Section 8.1 explains how this work contributes to the improvement and understanding of CNNs, in the meanwhile it also highlights the parts of this research that could be challenged or questioned the most. The limitations of structured receptive fields are discussed in section 8.2 and section 8.3 describes the things we would have done differently in hindsight. Finally, we will conclude this discussion by recommending the two most promising and interesting directions for future research on structured receptive fields.

8.1. CONTRIBUTIONS TO THE SCIENTIFIC WORLD

This section discusses the consequences of our findings for the research community. It describes how our findings contribute to the understanding and improvement of CNNs, how our findings should be interpreted and which parts could be questioned.

The main contribution of this research is a new idea to parameterize normal filters as a linear combination of gaussian derivatives. Thereby decoupling the structure and the scale of the filter: the structure is determined by the weights of the linear combination and the truncation order of the gaussian derivatives, while the scale is determined by the standard deviation of gaussian derivatives. The idea behind this particular parameterization is that the linear combination acts as a Taylor series expansion and therefore can approximate all structures learned by normal filters, while the optimization of the continuous scale parameter can serve as a proxy for optimizing the discrete filter size.

In this report we investigated both the ability of structured receptive fields to learn structure as well as their ability to learn scale.

First we address the ability to learn structure. In this report we give both theoretical evidence (appendix A) and empirical evidence (section 5.1) for the fact that structured receptive fields can approximate all filters learned by normal CNN filters. An example in section 5.1 however, shows that the quality of the approximation depends on the truncation order of the structured receptive fields. It is especially hard for structured receptive fields with a low truncation order to approximate normal filters that display high frequency oscillations (see figure 5.2). Finally, we note that empirical evidence is provided only for pre-trained filters of the AlexNet architecture. One could investigate whether our claim also holds for filters of other architectures.

Next we address the unique selling point of using structured receptive fields: they provide a natural way to learn feature scale. This report contains two experiments (section 5.2 and section 6.1) that show that structured receptive fields can indeed learn the filter size during the training stage. Both experiments are however performed on the MNIST dataset, and we would deem it wise to repeat these experiments in future research on more challenging datasets. For example, a dataset that consists of color images instead of grayscale images.

Finally, the experiment in section 5.3 demonstrated an advantage over using normal convolutional layers, by resembling a DenseNet architecture and replacing all the normal filters with structured receptive fields. The final result is a ~1% better test accuracy in favor of the structured convolutional layers. We reckon that this is only a single architecture, a single dataset and a single trainings run, and therefore it would not be wise to draw hard conclusions from this single result. Although it does demonstrate the practical potential of structured receptive fields, it is nowhere near something that we would call empirical evidence. Therefore one would need to repeat this experiment on many different architectures and datasets.

Unfortunately, we must also recognize a flaw in the way that this last experiment was conducted. The idea of this experiment was to replace normal filters with structured receptive fields, while keeping everything else the same. We failed to keep everything the same, because we did not regularize the structured receptive fields directly, but indirectly via the alpha parameters. One can show that L2-norm regularization of the alpha parameters is not the same as L2-norm regularization of the effective filters. For a fair comparison we should have regularized the effective filters directly, instead.

It is likely that repeating this experiment with the correct regularization, will change the outcome of this experiment. There are two possible outcomes: either the performance of the structured receptive fields becomes even better in which case it strengthens our conclusion that structured receptive fields seem to have an advantage over normal filters, or the performance becomes worse in which case we need to add to our conclusion that structured receptive fields require only their structure (the alpha parameters) to be regularized rather than the whole filter. The fact that filter structure and filter scale can be regularized independently of each other, might actually be another beneficial property of structured receptive fields. So repeating this experiment with a different form of regularizing will be very interesting, regardless of the outcome.

8.2. LIMITATIONS OF STRUCTURED RECEPTIVE FIELDS

There is only one real limitation of structured receptive fields in terms of what they cannot learn: structured receptive fields cannot approximate filters that contain a higher number of oscillations than their truncation order (see figure 5.2). This is because, gaussian derivatives have the same number of zero crossings as the order of their derivative. However, we would like argue that this restriction, is both a feature and a bug. It is a bug, because structured receptive fields cannot learn complex filter structures with a low truncation order, but it is a feature because we probably should not want to learn the complex structures as they are too specific and do not generalize well.

Another small limitation is that training an architecture with structured convolutional layers, takes approximately two times longer than training the same architecture with normal convolutional layers having 3x3 filters. This is mainly because structured receptive fields tend to learn larger filter sizes than 3x3, and therefore it takes more time to compute all the convolutions. Currently the gaussian basis filters are truncated at the very safe margin of 3 sigmas from the center, by lowering this factor one could decrease the filter size and speed up the training. However, one would have to investigate whether one is allowed to lower this factor without losing classification performance.

An other possible limitation might be the fact that, in the current implementation, all structured receptive fields in a convolutional layer share the same sigma. Therefore one would expect that approximating hundreds of normal filters simultaneously with a single sigma should be hard. However, the results in this report (section 5.1 and appendix B) do not seem to indicate that this is case. Possibly, because the all normal filters have approximately the same scale. After all, they all have the same filter size (i.e. 5x5 or 3x3).

The experiment in section 6.1 showed that the learned scale parameters can be used to change the scale at which the CNN operates, during the evaluation stage. This experiment only worked for upscaling of the images and not for downscaling. We also tried to downscale the MNIST test images and the scale parameters by a factor 2, but the final test accuracy dropped from 96% to 84%. This could either be, because the MNIST images are already quite small (28x28), but it could also be a fundamental limitation since one usually cannot lower the resolution of an image without losing information.

8.3. THINGS DONE DIFFERENTLY IN HINDSIGHT

The 'faster implementation' of the structured convolutional layer in section 2.5, was optimized for speed only. In hindsight we should also have focussed on memory consumption, because in the experiment in section 5.3 we could not fit a whole densenet architecture into GPU memory. Computational graphs in deep learning frameworks like Caffe, Torch, Theano, Tensorflow, etc., keep the input of every forward operation in memory in order to evaluate the derivative in the backward pass. For example, both the output (forward pass) and the derivative (backward pass) of the function $f(x) = x^2$, $f'(x) = 2x$ depend on the input x. Hence, it is important to use as little memory as possible.

The 'faster implementation' however, uses the Fourier transform to turn convolution in the spatial domain into element-wise multiplication in the frequency domain. Therefore, it also turns small spatial filters (i.e. 3x3) into large frequency filters (as large as the image). Therefore 'the faster implementation' requires much more memory than for instance the fast implementation. This and the fact that the densenet architecture itself is not very memory efficient (see [61]) makes that we could not fit the whole densenet in the GPUs memory during our experiment.

In hindsight, it might have been better to go for a different implementation that does not require so much memory. For example, an implementation that uses separable convolution. One could also reduce the number of multiplications by a factor 2, by noting that gaussian derivatives are always symmetric or anti symmetric. It is not clear whether this all, would have led to an implementation with a better speed-memory trade-off but at least, we would have kept it in mind if we could do it again.

8.4. RECOMMENDATIONS FOR FUTURE RESEARCH

In this section we discuss the two most promising and interesting directions for future research on structured receptive fields. The first one, is investigating whether replacing normal filters with structured receptive fields will always lead to strictly better or equal performance. The second one, is using structured receptive fields to investigate and possibly take over the role of subsampling layers.

This is a good moment to realize that structured receptive fields are nothing special, in

the sense that there is nothing that a structured receptive field can learn that a normal filter with a correctly selected filter size, cannot. Instead it is the intractability of manually tuning the optimal filter size for every convolutional layer, why using structured receptive fields should be preferred over using normal filters.

The experiment in section 6.2 showed, for example, that for normal filters poorly selected filter sizes can lead to a dramatic drop in classification performance and that this does not happen if structured receptive fields are used. It would therefore be nice, if one could show that the usage of structured receptive fields removes the need for an exhaustive search over all the possible combinations of filter sizes and that replacing normal filters always leads to strictly equal or better performance. In future research this hypothesis can investigated, by repeating the final proof of concept experiment on many different architectures and datasets that are commonly used. Even if structured receptive fields would only equal, or marginally improve the performance of normal filters, then they would still remove the need for this exhaustive search.

Moreover, there are also the extra applications that were briefly explored in chapter 6. These applications are unique to structured receptive fields, and provide valuable new insights into how CNNs, and subsampling layers in particular, work. The fact that structured receptive fields, like subsampling layers, can also increase the effective receptive fields without increasing the number of learnable parameters, makes them very suitable to investigate the role and the importance of the subsampling layer. In this research, we only briefly explored the possibility of using structured receptive fields for this purpose, but in future research this can be investigated in more detail.

It would be nice, if structured receptive fields could be used to discover the true role of the subsampling layer. Even more interestingly, once this role has been understood structured receptive fields might be able to take over that role and form an architecture that very elegantly, only consists of structured receptive fields. When designing new architectures, people would then no longer have to worry after which layer to include a subsampling layer.

The discussion above mentions the two most interesting ways to continue the research on structured receptive fields. We think that these directions cover important questions of which the answers could really make an impact. Furthermore we believe that the results in this report, already hint that these directions might be successful.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] A. M. Turing, *Intelligent machinery, a heretical theory,* The Turing Test: Verbal Behavior as the Hallmark of Intelligence **105** (1948).

[2] R. Szeliski, *Computer vision: algorithms and applications* (Springer Science & Business Media, 2010).

[3] D. Marr and E. Hildreth, *Theory of edge detection,* Proceedings of the Royal Society of London B: Biological Sciences **207**, 187 (1980).

[4] J. Canny, *A computational approach to edge detection,* IEEE Transactions on pattern analysis and machine intelligence , 679 (1986).

[5] C. Harris and M. Stephens, *A combined corner and edge detector.* in *Alvey vision conference*, Vol. 15 (Manchester, UK, 1988) pp. 10–5244.

[6] T. Lindeberg, *Feature detection with automatic scale selection,* International journal of computer vision **30**, 79 (1998).

[7] C. M. Bishop, *Pattern recognition and machine learning* (springer, 2006).

[8] K. P. Murphy, *Machine learning: a probabilistic perspective* (2012).

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

[10] D. Ciresan, U. Meier, J. Masci, and J. Schmidhuber, *Multi-column deep neural network for traffic sign classification,* Neural Networks **32**, 333 (2012).

[11] A. Krizhevsky, I. Sutskever, and G. Hinton, *Imagenet classification with deep convolutional neural networks,* in *NIPS* (2012).

[12] Y. Bengio, I. J. Goodfellow, and A. Courville, *Deep learning,* Nature **521**, 436 (2015).

[13] J. Neter, M. H. Kutner, C. J. Nachtsheim, and W. Wasserman, *Applied linear statistical models*, Vol. 4 (Irwin Chicago, 1996).

[14] D. W. Hosmer Jr, S. Lemeshow, and R. X. Sturdivant, *Applied logistic regression*, Vol. 398 (John Wiley & Sons, 2013).

[15] J. R. Quinlan, *Induction of decision trees,* Machine learning **1**, 81 (1986).

[16] C. Cortes and V. Vapnik, *Support-vector networks,* Machine learning **20**, 273 (1995).

[17] T. Cover and P. Hart, *Nearest neighbor pattern classification,* IEEE transactions on information theory **13**, 21 (1967).

[18] K. Fukushima, *Neocognitron: A hierarchical neural network capable of visual pattern recognition,* Neural networks **1**, 119 (1988).

[19] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, *Efficient backprop,* in *Neural networks: Tricks of the trade* (Springer, 2012) pp. 9–48.

[20] K. Hornik, M. Stinchcombe, and H. White, *Multilayer feedforward networks are universal approximators,* Neural networks **2**, 359 (1989).

[21] G. Gybenko, *Approximation by superposition of sigmoidal functions,* Mathematics of Control, Signals and Systems **2**, 303 (1989).

[22] K. Hornik, *Approximation capabilities of multilayer feedforward networks,* Neural networks **4**, 251 (1991).

[23] A. W. Roe, S. L. Pallas, Y. H. Kwon, and M. Sur, *Visual projections routed to the auditory pathway in ferrets: receptive fields of visual neurons in primary auditory cortex,* Journal of Neuroscience **12**, 3651 (1992).

[24] L. Von Melchner, S. L. Pallas, and M. Sur, *Visual behaviour mediated by retinal projections directed to the auditory pathway,* Nature **404**, 871 (2000).

[25] C. Métin and D. O. Frost, *Visual responses of neurons in somatosensory cortex of hamsters with experimentally induced retinal projections to somatosensory thalamus,* Proceedings of the National Academy of Sciences **86**, 357 (1989).

[26] J. Hawkins and S. Blakeslee, *On intelligence: How a new understanding of the brain will lead to the creation of truly intelligent machines* (Macmillan, 2007).

[27] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning,* Nature **521**, 436 (2015).

[28] Y. Bengio and Others, *Learning deep architectures for AI,* Foundations and trends{®} in Machine Learning **2**, 1 (2009).

[29] Y. A. LeCun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition,* IEEE **86**, 2278 (1998).

[30] J. J. Koenderink and A. J. van Doorn, *Representation of local geometry in the visual system,* Biological cybernetics **55**, 367 (1987).

[31] R. A. Young, *The Gaussian derivative model for spatial vision: I. Retinal mechanisms,* Spatial vision **2**, 273 (1987).

[32] J.-H. Jacobsen, J. van Gemert, Z. Lou, and A. W. M. Smeulders, *Structured receptive fields in cnns,* CVPR (2016).

[33] M. J. Tovée, *An introduction to the visual system* (Cambridge University Press, 1996).

[34] D. H. Hubel and T. N. Wiesel, *Brain mechanisms of vision* (WH Freeman, 1979).

[35] D. H. Hubel and T. N. Wiesel, *Receptive fields of single neurones in the cat's striate cortex,* The Journal of physiology **148**, 574 (1959).

[36] A. Witkin, *Scale-space filtering: A new approach to multi-scale description,* ICASSP (1984).

[37] J. J. Koenderink, *The structure of images,* Biological cybernetics **50**, 363 (1984).

[38] T. Lindeberg, *Scale-space theory: A basic tool for analyzing structures at different scales,* Journal of applied statistics **21**, 225 (1994).

[39] L. M. Florack, *Image structure* (Springer Science & Business Media, 2013).

[40] B. M. H. Romeny, *Front-end vision and multi-scale image analysis: multi-scale computer vision theory and applications, written in mathematica* (Springer Science & Business Media, 2008).

[41] T. Lindeberg, *Scale selection,* in *Computer Vision* (Springer, 2014) p. 701.

[42] Y. Liu, H. Li, J. Yan, F. Wei, X. Wang, and X. Tang, *Recurrent Scale Approximation for Object Detection in CNN,* arXiv preprint arXiv:1707.09531 (2017).

[43] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition,* arXiv preprint arXiv:1409.1556 (2014).

[44] R. K. Srivastava, K. Greff, and J. Schmidhuber, *Highway networks,* arXiv preprint arXiv:1505.00387 (2015).

[45] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition,* in *CVPR* (2016).

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

[47] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, *Densely connected convolutional networks,* arXiv preprint arXiv:1608.06993 (2016).

[48] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, *Reading digits in natural images with unsupervised feature learning,* in *NIPS workshop* (2011).

[49] A. Krizhevsky and G. Hinton, *Learning multiple layers of features from tiny images (MSc Thesis),* (2009).

[50] G. Verkes, *Receptive Fields Neural Networks using the Gabor Kernel Family (MSc Thesis),* (2017).

[51] S. Luan, B. Zhang, C. Chen, X. Cao, Q. Ye, J. Han, and J. Liu, *Gabor Convolutional Networks,* arXiv preprint arXiv:1705.01450 (2017).

[52] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, *Striving for simplicity: The all convolutional net,* arXiv preprint arXiv:1412.6806 (2014).

[53] Y. A. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, *Handwritten digit recognition with a back-propagation network,* in *NIPS* (1990).

[54] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift,* in *ICML* (2015).

[55] T. Lindeberg, *Scale-space for discrete signals,* IEEE transactions on pattern analysis and machine intelligence **12**, 234 (1990).

[56] B. Fornberg, *Generation of finite difference formulas on arbitrarily spaced grids,* Mathematics of computation **51**, 699 (1988).

[57] Y. A. LeCun, *The MNIST database of handwritten digits,* http://yann.lecun.com/exdb/mnist/ (1998).

[58] T. Cohen and M. Welling, *Group equivariant convolutional networks,* in *ICML* (2016).

[59] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, *Regularization of neural networks using dropconnect,* in *ICML* (2013).

[60] M. Lin, Q. Chen, and S. Yan, *Network in network,* arXiv preprint arXiv:1312.4400 (2013).

[61] G. Pleiss, D. Chen, G. Huang, T. Li, L. van der Maaten, and K. Q. Weinberger, *Memory-Efficient Implementation of DenseNets,* arXiv preprint arXiv:1707.06990 (2017).

V

APPENDICES

A

MATHEMATICAL PROOFS

This appendix provides mathematical proofs for some of the claims that are made in this report.

GAUSSIAN MODULATED TAYLOR EXPANSION

This proof demonstrates that structured receptive fields can be written in the same mathematical form as a taylor series expansion that is modulated with a gaussian envelope. Therefore any normal filter that is modulated by a gaussian envelope, can be approximated by setting the alpha parameters to the right values.

For convenience, the proof is given for a one dimensional structured receptive field. In two dimensions the proof follows exactly the same lines, but it requires keeping track of more terms.

Let $f(x)$ be a normal filter, then its Taylor expansion around x_0 is given by equation A.1. The taylor expansion is a power series where the coefficients β_n are determined by derivative of the normal filter in the point $x = x_0$.

$$f(x) = \sum_n \beta_n (x - x_0)^n, \qquad \beta_n = \frac{1}{n!} \frac{d^n f}{dx^n}\Big|_{x=x_0} \tag{A.1}$$

Next, the Taylor series is modulated with a gaussian envelope $G(x;\sigma)$ and expanded around the point $x_0 = 0$, see the left hand side of equation A.2.

$$G(x;\sigma) \sum_n \beta_n x^n = \sum_n \alpha_n \frac{\partial^n G(x;\sigma)}{\partial x^n} \tag{A.2}$$

A

We need to proof that a structured receptive field on the right hand side of equation A.2 can be written in the same mathematical form as the modulated power series on the left hand side.

The first step is to express the gaussian derivatives in terms of physicists Hermite polynomials H_n and a gaussian envelope $G(x;\,\sigma)$.

$$\frac{\partial^n G(x;\,\sigma)}{\partial x^n} = \left(\frac{-1}{\sqrt{2}\sigma}\right)^n H_n\left(\frac{x}{\sqrt{2}\sigma}\right) G(x;\,\sigma) \tag{A.3}$$

The physicists Hermite polynomial are defined by the following recursion relation:

$$H_n(x) = 2x\; H_{n-1}(x) - 2(n-1)\; H_{n-2}(x), \quad H_0(x) = 1, \quad H_1(x) = 2x \tag{A.4}$$

After substituting this expression into equation A.2, we get:

$$\sum_n \alpha_n \frac{\partial^n G(x;\,\sigma)}{\partial x^n} = G(x;\,\sigma) \sum_n \alpha_n \left(\frac{-1}{\sqrt{2}\sigma}\right)^n H_n\left(\frac{x}{\sqrt{2}\sigma}\right) \tag{A.5}$$

Next, the $\left(\frac{-1}{\sqrt{2}\sigma}\right)^n$ term is absorbed into the coefficient by redefining it: $b_n = \alpha_n \left(\frac{-1}{\sqrt{2}\sigma}\right)^n$

$$\sum_n \alpha_n \frac{\partial^n G(x;\,\sigma)}{\partial x^n} = G(x;\,\sigma) \sum_n b_n H_n\left(\frac{x}{\sqrt{2}\sigma}\right) \tag{A.6}$$

To proceed will use the fact that a weighted sum of polynomials is again a polynomial. Hence, we will unroll the sum over the Hermite polynomials, collect the terms and sum over the polynomials terms in x instead. For the purpose of illustration we unroll the summation for the first few values of N:

$$\sum_n^N b_n H_n\left(\frac{x}{\sqrt{2}\sigma}\right)$$

$$\begin{aligned}
N=0: &\quad b_0 \\
N=1: &\quad b_0 + \frac{\sqrt{2}}{\sigma} b_1\, x \\
N=2: &\quad (b_0 - 2b_2) + \frac{\sqrt{2}}{\sigma} b_1\, x + \frac{2}{\sigma^2} b_2\, x^2 \\
N=3: &\quad (b_0 - 2b_2) + \left(\frac{\sqrt{2}}{\sigma} b_1 + \frac{6\sqrt{2}}{\sigma} b_3\right) x + \frac{2}{\sigma^2} b_2\, x^2 + \frac{2\sqrt{2}}{\sigma^3} b_3\, x^3
\end{aligned} \tag{A.7}$$

From this it clear that a polynomial of degree N has emerged. We redefine the coefficients once more (see equation A.8), and sum over the polynomial terms to arrive at the same mathematical form as a modulated Taylor expansion, see equation A.9.

$$
\begin{aligned}
N=0: &\quad \gamma_0 = b_0 \\
N=1: &\quad \gamma_0 = b_0, \quad \gamma_1 = \frac{\sqrt{2}}{\sigma} b_1 \\
N=2: &\quad \gamma_0 = b_0 - 2b_2, \quad \gamma_1 = \frac{\sqrt{2}}{\sigma} b_1, \quad \gamma_2 = \frac{2}{\sigma^2} b_2 \\
N=3: &\quad \gamma_0 = b_0 - 2b_2, \quad \gamma_1 = \frac{\sqrt{2}}{\sigma} b_1 + \frac{6\sqrt{2}}{\sigma} b_3, \quad \gamma_2 = \frac{2}{\sigma^2} b_2, \quad \gamma_3 = \frac{2\sqrt{2}}{\sigma^3} b_3
\end{aligned}
\tag{A.8}
$$

$$
\sum_n \alpha_n \frac{\partial^n G(x;\sigma)}{\partial x^n} = G(x;\sigma) \sum_n \gamma_n x^n \tag{A.9}
$$

Finally, we note that the systems of linear equations that define the γ coefficients are consistent. Meaning that for any set of γ values there exists a set of α values that defines them. Thus, also when we choose the gamma values to be equal to: $\gamma_n = \frac{1}{n!} \frac{d^n f}{dx^n}\Big|_{x=0}$.

This concludes the proof that structured receptive fields can approximate any normal filter that is modulated with a gaussian envelope. Like a normal taylor expansion, the quality of the approximation is determined by the truncation order of the summation.

$$
\sum_n \alpha_n \frac{\partial^n G(x;\sigma)}{\partial x^n} = G(x;\sigma) \sum_n \frac{1}{n!} \frac{d^n f}{dx^n}\Big|_{x=0} x^n \tag{A.10}
$$

Q.E.D.

B

NORMAL FILTER APPROXIMATIONS

This appendix contains all the structured receptive field approximations to the filters from different layers of a pre-trained AlexNet.

Approximations to the 96 filters of the first convolutional layer are shown in the figures B.1 and B.2. In both figures structured receptive fields of order 4 are used, but in figure B.1 every structured receptive field has its own sigma while, in figure B.2 the structured receptive field approximations all share the same sigma. The odd rows show the original AlexNet filters and the even rows show the approximations. The sharing of the sigma causes the quality of the approximations to drop slightly, compared to the unshared sigma case.

Approximations to 16 of the 256 filters, in the second convolutional layer are shown in figure B.3. Because the number of channels is more than 3, it is no longer possible to show each filter in a single image. Therefore, each row shows a different filter and the columns display the different channels of the filter. The odd rows show the original AlexNet filters and the even rows show the approximations. The structured receptive field approximation share the same sigma.

Approximations to 16 of the 384 filters, in the third convolutional layer are shown in figure B.4. The structured receptive field approximation share the same sigma.

B

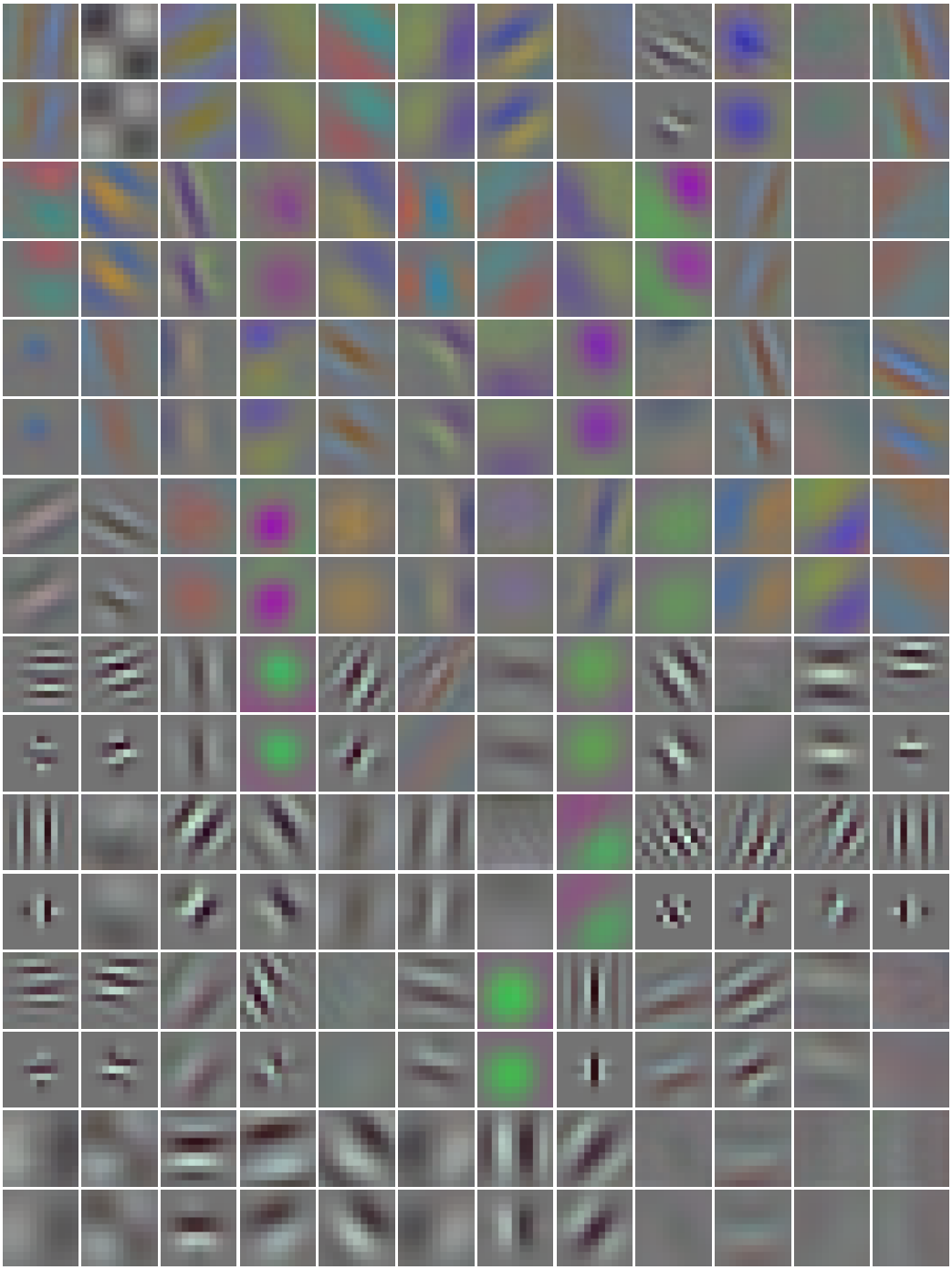


Figure B.1: Structured receptive field approximations of the first convolutional layer filters (11x11x3) of the AlexNet, with an unshared sigma. The odd rows show the original AlexNet filters and the even rows show the approximations with structured receptive fields. The approximations are obtained using structured receptive fields of order 4 that do not share the same sigma.

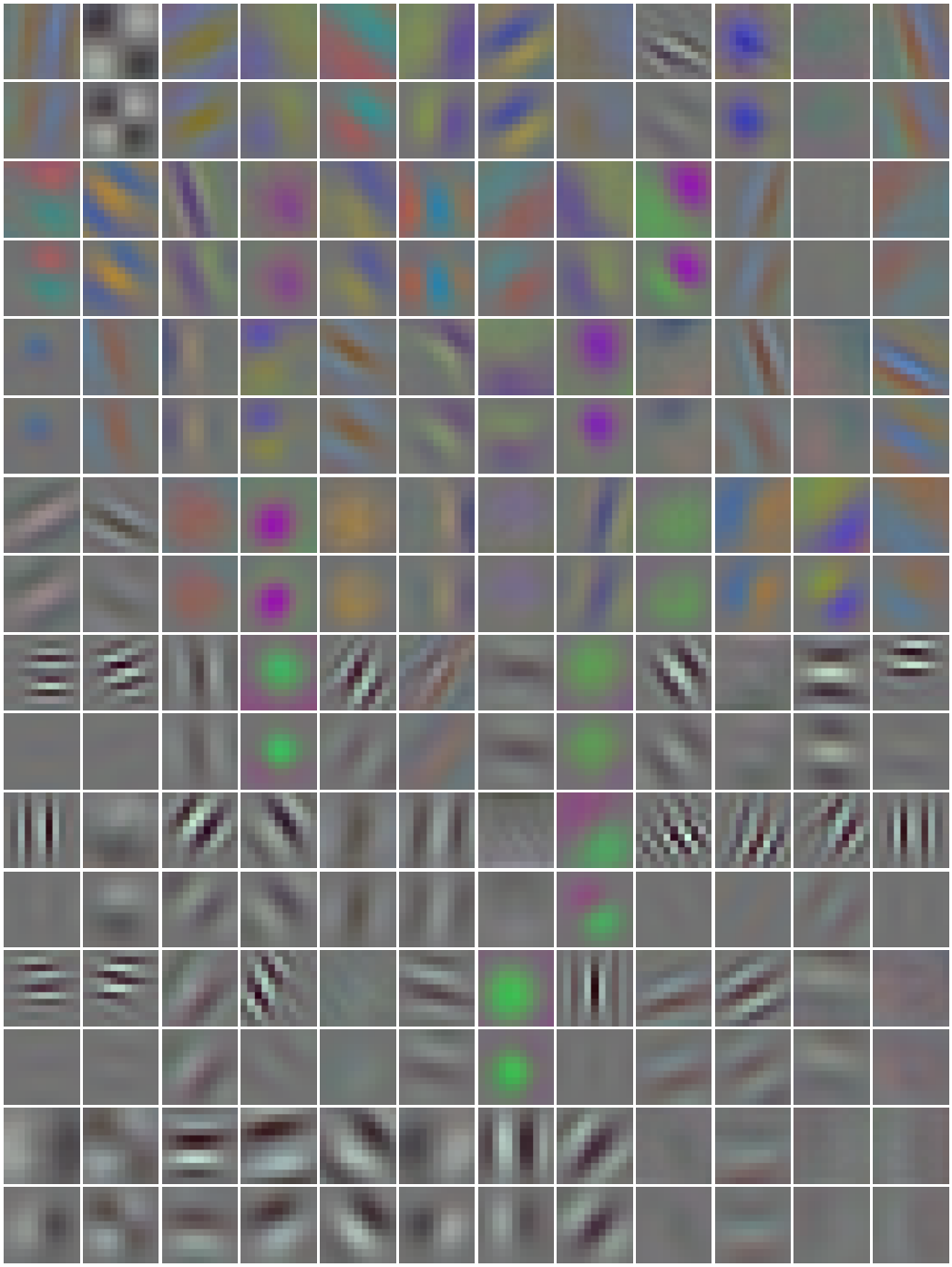


Figure B.2: Structured receptive field approximations of the first convolutional layer filters (11x11x3) of the AlexNet, with a shared sigma. The odd rows show the original AlexNet filters and the even rows show the approximations with structured receptive fields. The approximations are obtained using structured receptive fields of order 4 that do share the same sigma. The sharing of the sigma causes the quality of some approximations to drop compared to the unshared sigma case, see figure B.1. The final value of the shared sigma is 2.491.

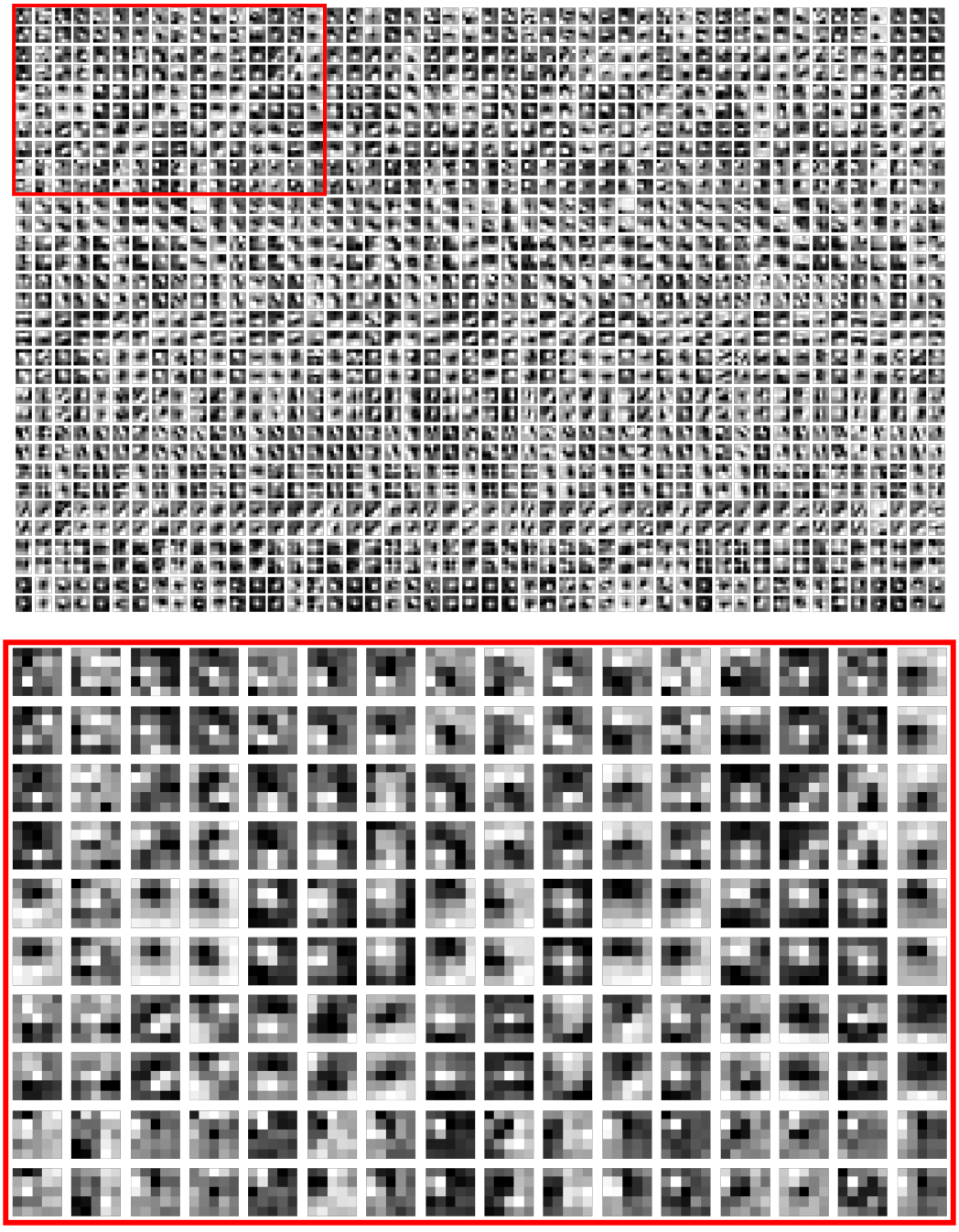


Figure B.3: Structured receptive field approximations of the second convolutional layer filters of the AlexNet (256 filters with dimensions 5x5x48). It is no longer possible to show the filters in a single image, because they consists of 48 channels. Hence, each row shows a different filter and the columns display the different channels of the filter. Because of the space, only the first 16 filters (of the 256 in total) are shown. The odd rows show the original AlexNet filters and the even rows show the approximations with structured receptive fields. The bottom figure is a zoomed in version of the red rectangle in the top figure. The approximations are obtained using structured receptive fields of order 4 that do share the same sigma (final value = 1.349).

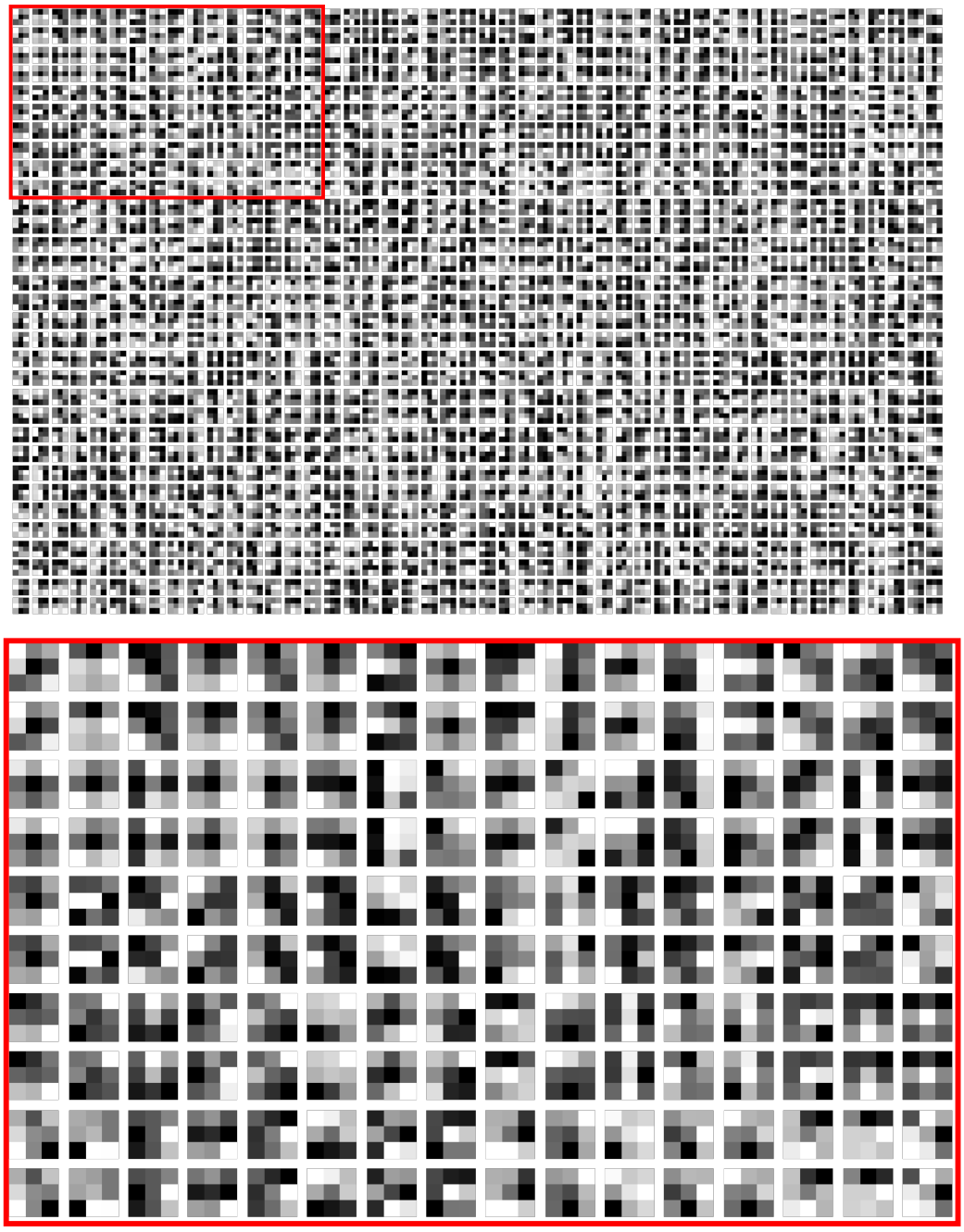


Figure B.4: Structured receptive field approximations of the third convolutional layer filters of the AlexNet (384 filters with dimensions 3x3x256). It is no longer possible to show the filters in a single image, because they consists of 256 channels. Hence, each row shows a different filter and the columns display the different channels of the filter. Because of the space, only the first 16 filters (of the 384 in total) and the first 48 channels (of the 256 in total) are shown. The odd rows show the original AlexNet filters and the even rows show the approximations with structured receptive fields. The approximations are obtained using structured receptive fields of order 4 that do share the same sigma (final value =). The bottom figure is a zoomed in version of the red rectangle in the top figure. From this figure, it is clear that structured receptive fields approximation is really close to the normal 3x3 filters!