University of Groningen

Gereon Vienken

Master's Thesis

Scale Selection in Convolutional Neural Networks with Dimensional Min-pooling and Scaling Filters

Supervisor:
Marco Wiering
(University of Groningen)

External Supervisor:
Sander Bohte
(Centrum Wiskunde & Informatica - Amsterdam)

August 28, 2016

university of groningen

faculty of mathematics and natural sciences

Abstract

Convolutional Neural Networks (CNN) are the most efficient method for image classification and recognition since the year 2012. In the last years they have been applied to more and more complex datasets and continue to hold the records in classification rates for most of them. But in order to perform well on bigger or more complex datasets, CNNs need to have more layers added to them. This makes them more computationally expensive and less biologically plausible. In order to counteract this trend, we propose a new type of convolution layer that makes use of multi-scale convolution filters, which are united with a MinMax function, in order to pass on the results of the most informative filter that is still noise resistant. In order to test this approach, we compare its results to a regular AlexNet implementation, as well as a multi-scale filter approach that uses Max-pooling. Our results show that MinMax can improve the performance of the standard network, even though the final results were not significant with ten iterations, because of overfitting in six of the networks. If we take this overfitting in to consideration and only look at the epochs before it starts, MinMax performed significantly better and improved the network performance by 0.71% on Cifar-10 and 4.9% on Places. This is in line with our expectations that MinMax has a higher impact on complex datasets with a lot of visual information regarding scale differences. These results can be seen as a first step and justify further research into this area, especially to test the impact of this approach when applied to deeper levels of the network.

Contents

Chapter 1

Introduction

The ambition to create a computational system that rivals humans in their mental and perceptual capabilities has been one of the fundamental driving forces behind the Artificial Intelligence (AI) community. Other researchers are taking a different approach, as they try to understand the workings of the human brain by rebuilding its components. They compare state of the art AI methods to humans in terms of their capabilities, as well as the types of errors they produce. This s possible because many state of the art techniques are modeled after what we know of human cognition. One of the main areas in which these two streams are working at the moment is the recreation of the human visual perception system. The invention of the Perceptron [19] was thought to have a major impact in approaching the workings of the human brain. Its ability to learn how to separate multi-dimensional data from examples by updating input weights seemed like a perfect simplification of the workings of a neuron in the brain. Sadly, its performance fell behind other classification methods of the time. This can mostly be attributed to the complexity of the most prominent problems. Image classification and their-like are notoriously difficult problems. Both the amount of information contained in one image and the possible information an image could contain, are nearly limitless. Furthermore, compared to the vastness of the information space, the differences between two categories can be very subtle. In order to make such a complex separation, multiple Perceptons need to be stacked in layers. This enables the Multi-Layer Perceptron (MLP) to model very complex separating functions but comes with the drawback that the network gets very big with a lot of weights, and these networks were practically impossible to train. We will go into more detail on this in section 2.1.1.
The invention of the Convolutionl Neural Network (CNN) [15] was met with skepticism. This was due to the previous failing of the Perceptron, as well as the inability of the network to outperform other approaches such as Support-Vector-Machines on the more complex datasets. It was not before

the publication of Krizhevsky et al, [14] that CNNs found traction in the larger community. In their paper they presented the results of their CNN on the ImageNet dataset, proving that CNNs were superior to other approaches, when provided with a large enough dataset and a powerful GPU that could train the network in a reasonable amount of time.
Currently CNNs still dominate benchmarks in visual classification challenges. They achieve this mostly not by changing the computations in the networks, but by making bigger and deeper networks. This trend was able to keep up with more complex datasets due to the fast improvement of GPU power in the last couple of years. However it seems that a lot of the insights we have gained into visual classification with other approaches or in neuroscience are going to waste, as they are not integrated in these state of the art methodologies.
In this paper we present a method which can be used in order to improve said networks. Our approach is separated into two parts. First we replaced the first convolution layer of a CNN with a scale-selection layer that uses different sized filters for the convolution. Second we replaced the normal spatial Max pooling over the output of one filter, with a MinMax algorithm that compares the same position from the output of different filters. Our objectives with this approach is to see if these changes can improve the performance of the network, compared to its original state. With this we attempt to make use of some of the knowledge acquired by neuroscience in the last decade, as well as ideas used in the computer vision debate, before Convolutional Neural Networks dominated the stage.

1.1 Research Question

In order to increase the performance of CNNs on more complex and bigger datasets the networks are constantly increased in depth. This becomes apparent when we compare an older architecture like the AlexNet, which won the Imagenet challenge in 2012 with the newer MSRA architecture that won in 2015 (Figure 1.1). But we do not see such deep structures in the brain. There seem to be different methods used in the brain we are not yet aware of that help the brain structuring data in a more efficient way than CNNs are currently capable of. One of these methods could be an internal scale selection method that helps to constrain the network. In this thesis we want to explore whether such a method can be implemented in a CNN, and whether or not this helps a simple network to perform better on a more complex dataset.

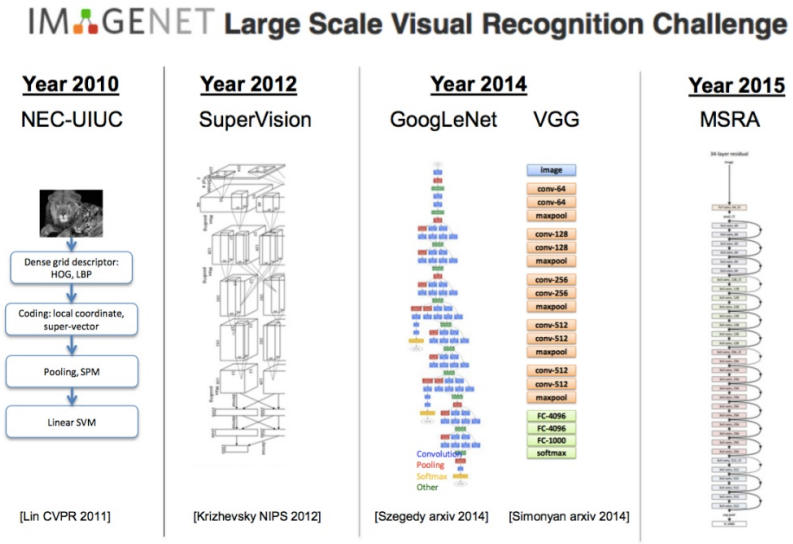


Figure 1.1: Winners of the ImageNet Challenge with network depth.(Taken from http://www.slideshare.net/nmhkahn/case-study-of-convolutional-neural-network-61556303)

Chapter 2

Theoretical Background

2.1 Convolutional Neural Networks

2.1.1 Neural Networks

Neural networks are a classification algorithm that comes closest to being a simplified model of our understanding of the human brain. Neural networks are clusters of smaller calculation units called Perceptrons [19]. Perceptron is a term used to describe an algorithm that combines weighted sums with some non-linear transformations (See figure 2.1). By gradually adapting the weights whenever the computational result does not match with the expected results, the Perceptron slowly learns to differentiate between the different classes and becomes a separator for the learned classes.
By combining different Perceptrons in the neural network, we effectively create a nonlinear classifier that can structure separable datasets of unlimited dimensionality, given the network is big enough and sufficient training data is provided. This theoretical concept however runs into multiple problems when applied. Since the structuring of complex data requires Perceptons to be stacked on top of one another, the network gets ever "deeper". But there are limiting factors concerning this depth. First of all, every connection in the network requires a weight from every neuron of the previous layer in order to be fully connected. For deep Neural Networks this amount of weights and computations can become so large that they are not computable on regular machines. Secondly, with every layer of depth, the error propagated back in a network decreases, leaving only very small gradients for the beginning layers to adapt with. This problem is usually referred to as 'vanishing gradients' [9]. Because of these problems neural networks had fallen out of favor during the 1990s. It took until the year 2006 for new methods to emerge or being rediscovered, more available data and better technology, for neural networks to return to popularity.
The network type that emerged and dominated most vision related challenges was the Convolutional Neural Network [12]. Convolutional Neural

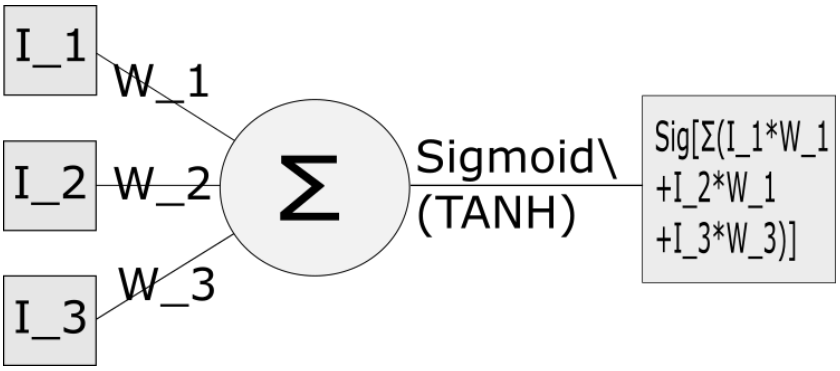


Figure 2.1: A basic Perceptron. The output of the perceptron is calculated by applying the sigmoid function to the sum of all the products of the Inputs with their corresponding weights.

Networks (CNN) differ from basic Neural Networks (NN) by their architecture, as well as the computation in the different layers. Regular NNs consist of an input layer, any number of hidden layers and an output layer (Image 2.2).

CNNs consist of multiple modules that are always followed by a fully connected NN at the end. The modules are built up of different layers. Each module has at least a convolutional layer and a non-linearization layer. These can be followed by pooling layers at some stages of the network (Image 2.3). These modules can occur in a wide variety of iterations and parallelizations, depending on the architecture and implementation. The following sections will explain these layers in detail.

2.1.2 Convolution

Convolutional Neural Networks (CNN) are a particular type of Neural Networks that have replaced lower layer Perceptrons with a slightly different calculation, when applied to vision problems. As the name suggests instead of fully connected Perceptrons with a weight for every input unit, their weights are used as a form of convolution-filter that is convolved with the input. To illustrate this, we take a theoretical network. Node A of the network has as input a 32x32 pixel image. In a regular Neural Network that one node would have 32*32=1024 connections to the input with the same number of weights. In a CNN, node A could have, for example, only 9 weights. These 9 weights would be arranged in a 3x3 filter, which would then be convolved with the image to create a new output image.
This is done by multiplying the A pixel value from the image with the corresponding value in the filter. Then all the values are summed up to create the new pixel value for the output (See figure 2.4). Sliding this filter over

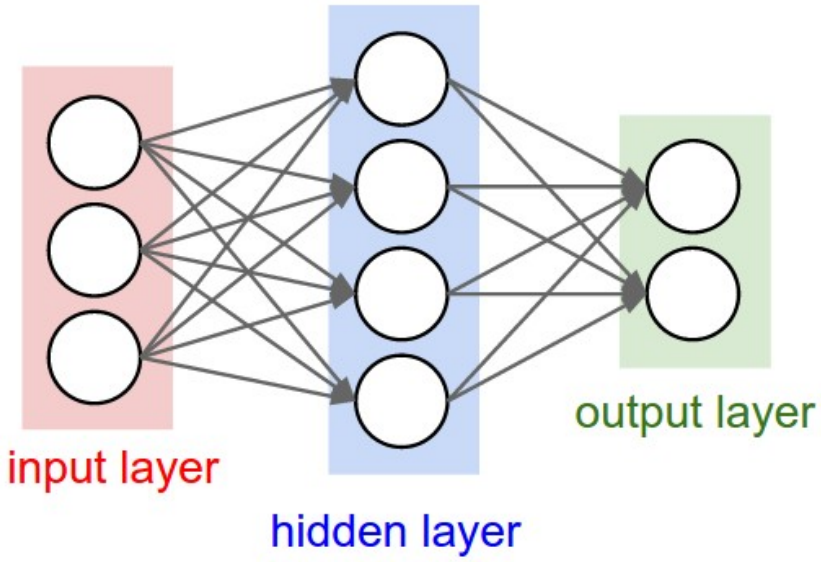


Figure 2.2: The Basic Neural network architecture, consisting of multiple layers of Perceptrons. Taken from http://cs231n.github.io/neural-networks-1/

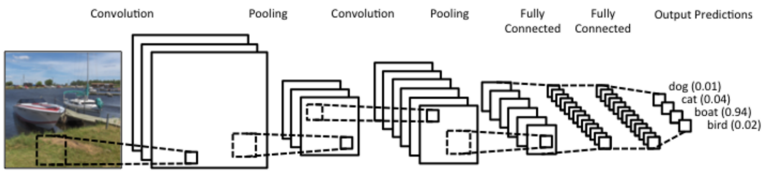


Figure 2.3: The Basic Convolutional Neural network architecture, consisting of multiple convolutional, pooling and fully connected layers. Taken from http://d3kbpzbmcynnmx.cloudfront.net/

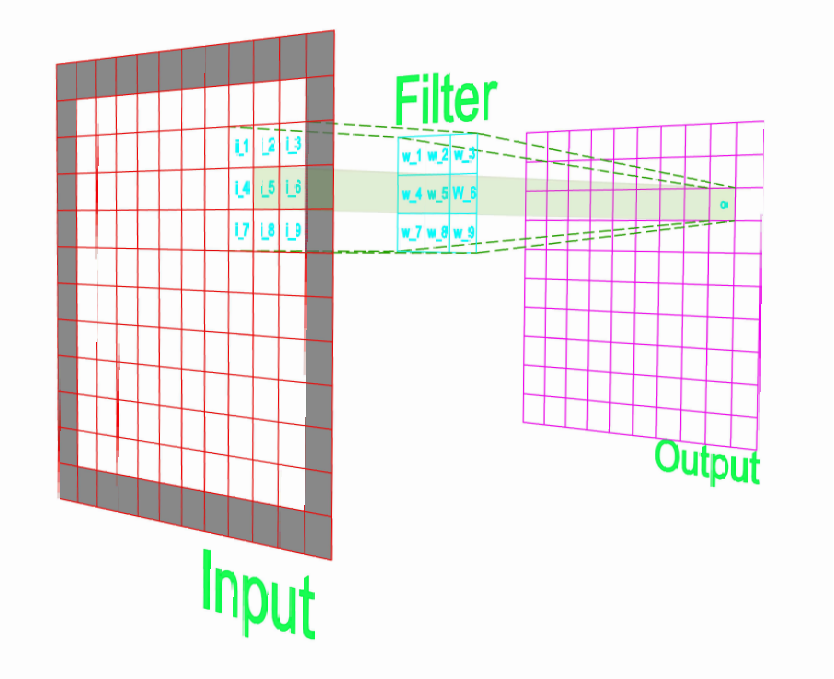


Figure 2.4: Convolution of Filter and input. To calculate Output value o we use $i_1 * w_1 + i_2 * w_2$...

the entire input image will create the corresponding output image. In order to get output of the same size as the input, many functions make use of a process called "zero-padding". For this the image is surrounded with rows and columns that contain only "0". For the output to be same size as the input we need $(Filtersize - 1)/2$ lines added on each side. Every pixel of that output would then go through a non-linear transformation, before it is passed on to the next layer.
Newer CNNs add different steps in between, mostly a method of pooling, that reduces the size of the output before it is passed to the next layer. More information about this process can be found in section 2.1.4. Nearly all convolution layers consist of more than one node and the output images of all the nodes are passed to the next layer in parallel.
On top of these convolution layers is a fully connected Neural Network, that would perform the actual classification task, based on the features that were extracted by the filters.

2.1.3 Non-linearisation

The non-linear transformation has also changed over time. In the past, for Perceptrons/neurons mostly functions like Tangens Hyperbolicus or Sigmoid

were used. These led to the vanishing gradients in deeper networks and have therefore been replaced. Most noticeably in non-linear transformations in CNNs nowadays are the Rectified Linear Units (ReLU), which use the simple operation:

$f(x) = Max(0, x)$

This improved the back propagation because essential for updating of the weights is the derivative of the activation function. For ReLU this simply gives:

$$\frac{\partial f(x)}{\partial x} = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}$$

which means the error can just be fully propagated for every value bigger than 0 and is not updated otherwise. But even if this helps with the vanishing gradients, it creates a new problem which is commonly called "dead units" or "dying ReLU". This describes the fact that a very large gradient can lower the weights on some connections so far that they never get activated and therefore never get updated. But this can be overcome by a simple method called "leaky ReLU". In leaky ReLU, the values get updated with:

$$\frac{\partial f(x)}{\partial x} = \begin{cases} 0.01x & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}$$

This means that even inactive units get updated by a small margin, which lets them recuperate after their values have dropped.
A newer activation function that has been introduced in 2016 is the Exponential Linear Unit (ELU). ELU behaves the same way as ReLU for values bigger than 0. But for values smaller or equal they use an exponential function:

$$f(x) = \begin{cases} \alpha(\exp(x) - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$$

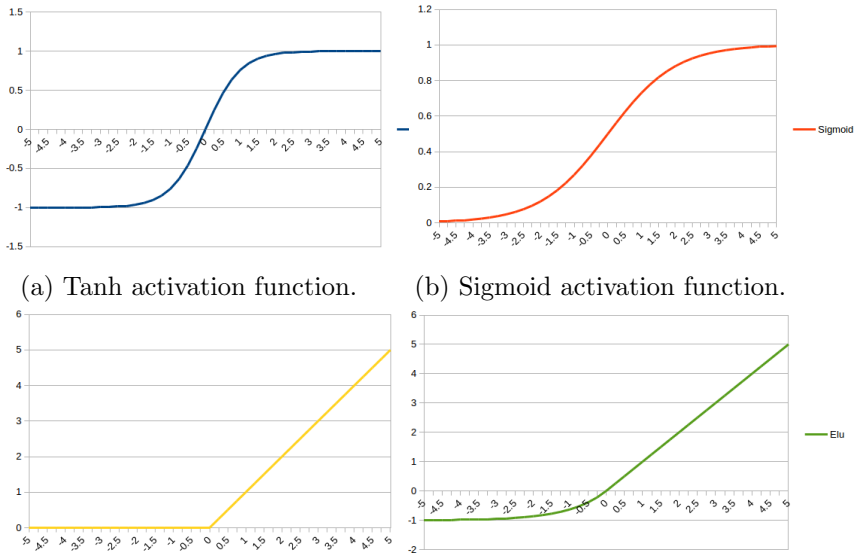
Here α is a control parameter that describes the curving of the function. In contrast to ReLU, ELU is capable of producing negative values. This increases the performance and decreases the chance of overfitting. For the back propagation we get the simple derivative of:

$$\frac{\partial f(x)}{\partial x} = \begin{cases} f(x) + a & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}$$

Since this derivative has no 0 for highly negative numbers we do not run into the "dead Unit" problem and research has shown that it also has a positive effect on the network in terms of over-fitting.

2.1.4 Pooling

After the non-linear transformation some CNNs have a pooling step in some layers. There are different forms of pooling, but at the moment the most noticeable ones are Average- and Max-pooling, where Max-pooling (Figure



(a) Tanh activation function.

(b) Sigmoid activation function.

(c) Relu activation function.

(d) ELU activation function

Figure 2.5: Activation Functions

2.6) is gaining more popularity. Pooling works essentially in the same way as the convolution-layer. The difference is that in the pooling all weights are 1 and instead of summing up all units in the filter we average or take the maximum. In the pooling layer this mostly means that the step size is bigger than one, so that the resulting image is a condensation of the information in the input image. This step improves the scale and rotational invariance of the network since the information of broader areas is combined in smaller ones while they are filtering out noise. This also improves the over-fitting rate of the network [22].

In order to avoid confusion, we note at this point that there is a differ-

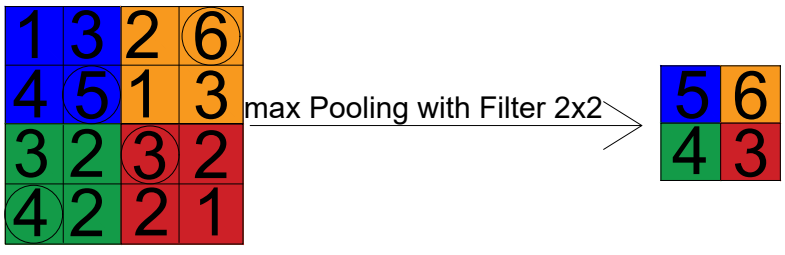


Figure 2.6: Max-pooling with Filter-size=2 and Step-size=3

ence between the Max-pooling mentioned here and the Dimensional-Max-pooling, that is described further in this thesis. The Dimensional-Max-pooling (DMP) or Competitive-Max-pooling does not refer to the process described above, but a process that compares the values of different output images. In DMP we compare the values of pixels at the same positions over different images. So while Max-pooling produces one output image for one input image, that is smaller than the input image, DMP produces one output image with multiple input images, where the size of the output image matches the size of the input images.

2.1.5 Fully Connected Layer

As mentioned before, the actual classification of the network is done by a fully connected Multi Layer Perceptron.

2.2 Filter Scale Selection

Scale difference is a problem which occurs in most types of image recognition processes. In this context we do not directly refer to the difference in scale, in which the same object can occur in an image, but a sub problem that can occur in images when differences in scales are present. The scale we are talking about, is the scale of the optimal filters used to extract features from the image. For the most parts this would be edge detection filters. The problem here is that features in an image can vary dramatically in their clarity and contrast. This can be a result of lighting, the focal length of a camera lens, the coloring of objects or noise in the image. These differences can lead to issues for edge detection programs, because an algorithm that excels at enhancing very subtle edges may also be very susceptible to noise and create edges out of slight distortions in the image, that do not represent edges in the real world. Whereas the opposite algorithm, that can filter out noise, might filter out details of the image as well. This is illustrated in image 3.2. Most edge detection algorithms either fall in one of those categories or try to strike a reasonable compromise between the two. The idea used in this thesis, however, is built on the idea of Zucker and Elder [6], described in the next section, which aspires to get the best from both worlds.
In Convolutional Neural Networks the problem of different feature scales is handled to a certain degree. Since the filters in every convolution layer converge to the filters that hold the most information for the training set it is entirely possible that a network with a large number of filters per layer can get different filters that recognize the same feature on different scales of clarity. However, providing the network with a large enough number of filters to do so is not practical.

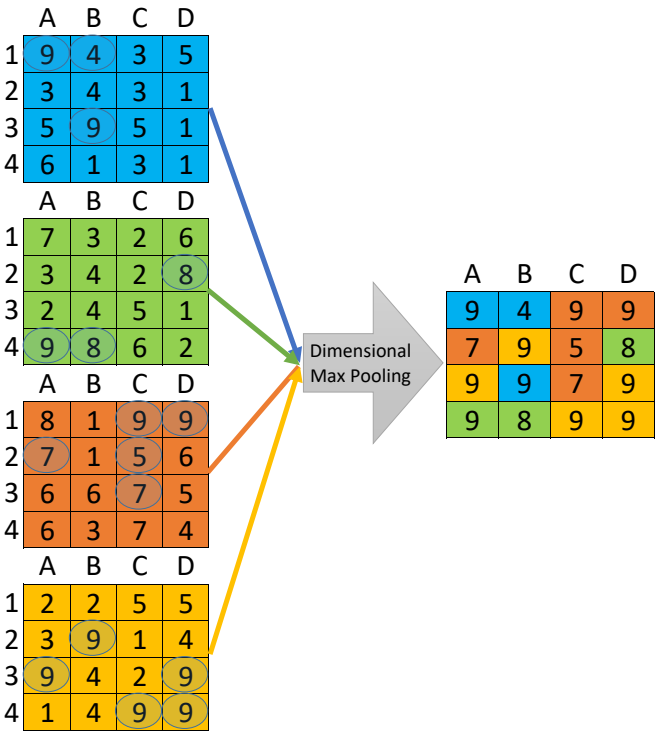


Figure 2.7: Dimensional Max Pooling. The output consists of the maximum values for all positions.

Chapter 3

Related Work

3.1 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are as of now the leading method in most complex recognition tasks, including image classification [14], voice recognition [8], speech recognition [1], gender-recognition [24] and video classification [11].
CNNs, in large parts, owe this success to the fact that its nodes operate as feature detectors [2]. These features become more and more sophisticated in deeper levels of the network [29], which give CNNs the ability to make specific generalizations even with a large number of classes to be distinguished. This kind of hierarchical structuring closely resembles the architecture that has been found in the Visual Cortex of the brain [10], where simple and complex cells have a similar relationship as the low and higher level filters in a CNN. That is not to say that CNNs are a perfect model of the human visual system. Research has shown that CNNs can make mistakes that we do not see in humans. Nguyen et al. [17] have shown that it is possible to craft images to fool CNNs that could never fool a human, while Szegedy et al. [23] showed that slightly altered images that make no difference to humans in terms of recognition can lead CNNs to miss-classify.

CNNs have also been used for other things aside from the typical recognition and classification tasks. Based on a paper by Simonyan et al. [20] where they showed visualizations of learned attributes in a CNN, a whole artistic style has emerged in which images are passed through a CNN similarly to an autoencoder, so that the network pushes its learned features on the image, creating an abstract mixture of the image with the concepts learned by the CNN. This can for example be used to train a network on a specific artistic style to then "impose" that style on an input image [7].
Even though the different filters in different layers in a CNN can have different sizes, most standard CNNs like AlexNet [14] or VGG [21] have only

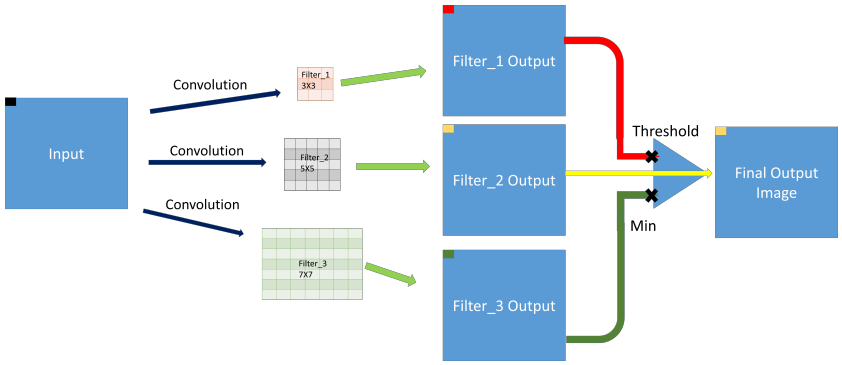


Figure 3.1: MinMax-module computation. The input is convolved with the convolution filters. From those results the pixels in the same position are then compared. For every position the value of the pixel is passed through that was created from the smallest filter whose value exceeds the threshold.

one size of filter per layer. But in newer networks the idea of multiple scaled filters per layer has emerged. Most noticeably in the very successful Inception models [22], where the outputs are simply concatenated, or a model where the filters are combined with competitive Max-pooling [16]. A fact that needs to be mentioned is that our implementation makes use of different scaled filters and even though they are initialized as scaled versions of one another we did not train them dependently. This means that every filter after being initialized has the ability to change away from its scaled versions. This decision was made for multiple reasons.
First it does not appear to be biologically plausible for the brain to have an intrinsic mechanism that keeps different cells to react to scaled versions of the same input. To our knowledge there has not been any study that found such a mechanism. The fact that the brain has cells that still seem to do this, possibly indicates that the learning mechanism itself facilitates this structure. Therefore we were interested to see if this behavior would emerge in our network.
Secondly, it has been shown that correlative filters themselves can improve network performance [4]. As such we would need to make sure that any result would not be attributable to this mechanism and therefore we decided to not use it for this pilot study. Never the less this possibility should not be disregarded for future research.

3.2 Visual Information

In order to recognize or classify images correctly, there are different approaches one can take. Problems that consist of comparing a given example with single instances in a database, like face recognition or fingerprinting

mostly make use of features that were pre-implemented by humans in order to help the algorithm to distinguish between the data. This could be symmetric features in the face that best distinguish facial structures, or prominent patterns in fingerprints, such as arcs and spirals. In classification tasks however this is not an option, because a dataset might contain classes from which some are very similar and some are not similar at all. Hand crafting distinguishable features that can be used to recognize classes reliably, for datasets with thousands of classes, is therefore not feasible. Convolutional Neural Networks can solve this problem by making use of the fact, that complex distinguishable higher level features are composed of unique combinations of simpler features. This hierarchy from simple to complex features can be found in the filters of the CNN, where first layer filters mostly consist of Gabor-like structures, while deeper in the network filters contain features that can be recognized as objects of the world.
An important part of the processing of visual information is the recognition and enhancement of edges. It has been known for a long time that edge enhancement takes place in the eyes themselves, even before the information reaches the brain, which processes it. So every AI algorithm that draws inspiration from, or tries to emulate the human brain, should take a good look at this process. Edge detection has been a research topic since the beginning of image processing and a lot of different approaches have been used with varying degrees of success. Surprisingly an approach used, even before CNNs is the so called "Scale-space technique" by Witkin [27]. This technique consists of convolving the image with a Gabor-filter in order to enhance edges. A process, that is nearly the same as the first stage of a CNN, considering that the first filters in a CNN mostly closely resemble Gabor-filters. The method on which we based this research was also originally designed as an improvement to the scale-space method.
To train and test implementations that try to analyse visual information, image datasets are needed, that fulfil certain criteria. The most important criteria is that the classes can only be distinguishable by features that are a part in the class. This means that there cannot be a distinguishing feature in the image that is not part of the object. For example, when distinguishing between animals a network could learn that every image for the "Bird" class has a blue sky as background, whereas none of the other animals does. If this were the case, the network would default to these easy distinguishable criteria to identify birds, but would not be able to identify a bird that is not depicted in front of that blue sky. In order to make sure something like this does not happen it is necessary to have a large amount of different images for the algorithms to train on. Commonly used datasets that have this criterion are for example the ImageNet dataset [5] or the datasets used for this research: Cifar10 [12] and Places [30].
The ImageNet dataset was the first dataset on which CNNs officially outperformed every other method in 2012 [8], and they have continued to do

so since then [22].

3.3 Dimensional Min-Pooling

The main idea that this thesis brings to the scientific debate is the use of Dimensional Min-pooling for Multi-Scale Convolution-Filters. The idea goes back to the paper "Local Scale Control for Edge Detection and Blur Estimation" by Zucker and Elder [6]. In their publication they argue that the best way to detect edges in an image with a broad range of blurred scales, would be by using a "minimum reliable scale" that locally estimates the blur over the image. This is necessary because in real world images edges cannot always be described as a simple step discontinuity. Different properties, of both, the light source and the structure of the reflective surface can lead to a disruption of the edges in the resulting image. The examples they give focus on the lens related depth-of-field, the penumbral blur of an object's shadow, as well as the shadow of a rounded edge. All of these effects can lead to highly different blur scales over the image. To illustrate this, they used the image 3.2 of a mannequin illuminated from behind. The mannequin itself shows a high amount of detail, while the shadow becomes blurred over distance. This contrast makes it impossible to find a reliable spatial scale that can detect all the details of the mannequin, while reliably finding the shadows' edges without producing noise. The solution they presented for this is the minimum reliable scale. This refers to the most accurate filter size which is still reliable in filtering out the expected noise in the image. This is done by calculating the second derivative of the intensity function at every point in the image to find the zero-crossings, which are used to identify the edges.

The idea was then picked up by Bohte, Scholte and Ghebread in 2015. At this moment their results are not published, but these unpublished results form the basis of this thesis. They show, that for local scale selection the results for Minmax over multiple scale filters approximate the results of the most reliable scale for most of the tested groups of natural images and outperform simple Max.
From this idea we created our MinMax module (Figure 3.1). The module takes the outputs from the convolutions of the input image with different scaled filters. From every output image produced this way, we compare the pixel values of the same position. For each of those positions, the value created by the smallest filter, that exceeds the threshold value is passed on. We do this because in smaller filters the value of each pixel has more impact on the output value, leading to a higher resolution. The threshold value is in place to give an indication when the information is graded, in which case

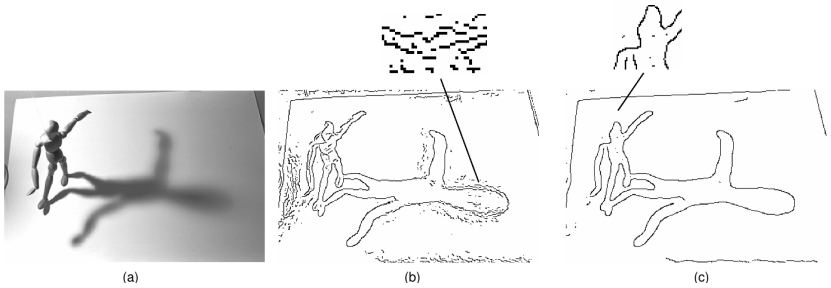


Figure 3.2: Illustration of the local scale estimation problem, published by [6]. Different edges in the image (a) need different spatial scales in order to detect their edges reliably. The First (b) algorithm detects the edges in the mannequin reliably but is over sensitive to the blurbed edges of the shadow. The second (b) algorithm detects the edges of the shadow reliably, but is not sensitive enough for the details of the mannequin.

bigger filters can better filter out the noise in individual pixels. The exact implementation can be found in chapter 4.

3.4 Filter Scale Selection

The approach we propose to improve this process tries to find the optimal scale at every pixel. A detailed description of the process can be found in Section 3.3. For the training of the filters we decided to let the filters train independently of one another. It might seem counter intuitive, but this was done in order to allow the network to learn the most information it can with the given number of filters. We also considered a setup with dependently trained filters but did not use it for this thesis because of the literature on which this thesis is built. Liao and Carneiro [16] used an implementation that utilized multi-scale filters, that were combined with a max-pooling algorithm. They referred to what we call Dimensional-Max-pooling, as Competitive-Max-pooling, because the max-pooling, that combines the different filters, has the different subnetworks, that lead to the input images, compete for providing the final output of the layer. The Competitive Max-pooling algorithm they described, showed that a competitive-max-pooling in the now popular inception model, outperforms the basic inception model [22]. These two algorithms and the theory behind them might differ from our own but it shows that different ways of approaching a problem can still lead to similar applications.
A major difference between our approach and those is that inception style networks make use of multiple sized filters on every convolution step of the network. Since our idea is based on papers that describe general edge detection/enhancement solutions and have no relation to the higher level features

that are normally found in deeper level filters of a CNN we decided to limit our approach to the first layer of filters and use single-scale filters for the deeper levels. It would have been possible for us to test this approach with parallel trained filters which would have been closer to the original idea. We decided to try and stay between the two. This has two reasons:
Firstly, this is a Master Thesis that should give an indication if the idea can be of general merit to the CNN discourse. If our network indeed performs as expected, this would be an indication that further research in this aspect could be worth pursuing and since our method is generally applicable, it could also be applied to an inception model for further study.
Secondly we wanted to show that the idea of MinMax, as a more biologically plausible idea, can be translated to a neural network setting. Since the approach has only been used for general edge enhancement, outside a network setting, we can still not be sure about this. We could prove this however, if our network could outperform a normal network of the same size. Furthermore, if the filter in the layer would converge to scaled versions of one another, this could give some explanation why we find such functionalities in the human brain.
A different possibility that should also be mentioned here is to let the filters train dependently but not keep them strictly scaled versions of one another. Research in this direction has been done by Chen et al. [4]. They used dependently trained filters but in contrast to what we need, they had their dependent filters reverse correlated so that they would correspond to opposite features. They do not achieve this by defining one feature as the opposite of another, but by defining one filter as the master filter and the other as dependent. Then both get updated, by not just the partial derivative of their own cost function, but also the negative partial derivative of the other parties cost function. This also allows for multiple filters to be dependent on one master filter. This could be a usable approach for future alterations of our function. We would only need to find a suitable way to relate the filters. This could for example be done by using the largest filter as master and the smaller filters as dependent. The filters would then be updated in a similar manner, but with the error matrices, scaled to their respective sizes. The problem with this is that in our implementation only the selected filter gets updated for every pixel. This leaves most of the error matrices with 0 which could pose a problem especially since the original implementation used averaged values for the dependents. This should be thoroughly researched, before put into the model.

Chapter 4

Method

4.1 Torch

Torch is a free machine learning framework for which libraries for deep learning and CNNs exist. The main programming language is LUA, while the underlying calculations can be done on a GPU with Cuda. With the CNN implementations provided by Soumith[1] it is possible to get a running implementation of one of the standard CNNs like Alexnet or VGG on GPU, for the imagenet dataset. It is also relatively simple to make changes to the network by changing the parameters or use some of the available modules. A drawback of this library is that the modules are written in LUA, a not very common programming language. Furthermore, making changes to the actual calculations behind the modules requires changes in the underlying C or Cuda code. For programmers who are not familiar with these languages, but who need to create their own modules, it could be beneficial to make use of one of the free deep learning libraries available, like Theano or Tensorflow in Python or the Matlab deep learning library.

4.2 Networks

The implementation for this project consists of a scale selection module that replaces the first layer of a convolution neural network. For testing we made use of the AlexNet implementation from Soumith[2]. We replaced the first convolution filters with a parallelized setup of convolution-filters, that are separated in four different sizes. Every filter in one size is initialized as a scaled version of the filters in other sizes. The results of the convolutions are then passed on to the MinMax algorithm which decides for every set of scaled filters which value is passed on for every pixel. The output of the MinMax function consists of the output image where every pixel value is

[1] https://github.com/soumith/imagenet-multiGPU.torch
[2] https://github.com/soumith/imagenet-multiGPU.torch

constructed by the most accurate filter size.
The scale selection module is made out of multiple convolution filters of different sizes. In order to compare the results of our approach with a standard AlexNet implementation on both datasets we modeled the AlexNet setup to previously described setups on the individual datasets and then replaced the first layer with a number of convolution filters that would sum up to a number, close to the number of filters in the original network.

4.3 Min Module

The main objective of the proposed implementation is the selection of the optimal scale for the filters. In order to achieve this, we utilize the MinMax operation previously described in section 2.2. We implemented the MinMax algorithm using two separated parts. The first effectively implements the convolution filters, the second part takes care of the actual MinMax- pooling.

4.3.1 Multi-scale Filters

The implementation of the multi-scale filters was based on the "SpatialConvolution" Module provided by the "nn" library of Torch. In order for our filters to be initialized as scaled versions of one another we added a new variable to be passed to the node upon creation. The variable consists of its "mother-node". A node that was passed this variable would not create its own filters, but load the filters of its mother-node and scale them to the size it was given. We decided for our implementation to use the node with the biggest filters as the mother-node, in order to ensure that we would not lose any information because of upscaling, but could simply downscale without any losses.
As explained in section 2.2, we decided to go with a simple filter training as described in section 2.1.2 instead of using scaled versions of identical filters for all sizes.

4.3.2 MinMax

The implementation of the MinMax module was based on the code of Bohte et al.'s unpublished paper where they implemented an enhancement filter in Matlab[3]. Their code used pre-implemented absolute Gabor-filters. They first calculated the convolution results of all filters. Then for every pixel, the value, which was generated by the smallest filter given that the result surpassed the threshold, was selected as the final value for this pixel. The threshold was determined by calculating the value distribution of the output. Every value smaller than two standard variations would be considered below

[3] https://de.mathworks.com/campaigns/products/ppc/google/matlab-b.html?s_eid=ppc_29743045882&q=matlab

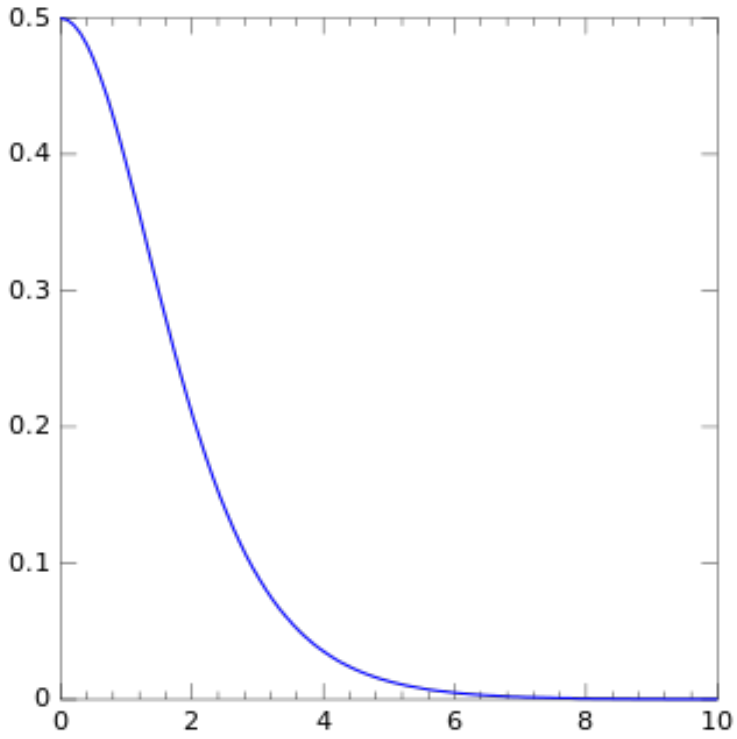


Figure 4.1: A Half-Gaussian distribution is a normal distribution folded at the mean.

the threshold and disregarded.
But his method was based on three assumptions:

1. The input images have all positive values
2. The filer values are independent of the threshold.
3. The value distribution of the convolution-output follows a Half-Gaussian distribution (See Figure 4.1).

We can already see that the first two points do not hold for our CNN. For the original implementation the filters were pre-made and did not change after being introduced to the system. But that is not the case in a CNN. Since the filters are initialized at random and trained with back propagation, the decision made by the threshold function has an impact on the filters.
We also choose not to convert all values into positives, either by taking the absolute value or adding the smallest negative value to all values. Even though we tried to use all positive input images, this had an adverse effect on the classification results. Therefore we decided to go with the standard input normalization, as determining the exact influence of changes in the

input was not in the scope of this project.
Even the third assumption does not work for our implementation, as can be seen in figure 4.2 where we look at the value distributions of our filter output. The images depict the value distribution of the pixels for two different filters each in 4 different sizes (Nr.1 - Nr.4). Histograms labeled with Nr.1 are the pixel value distribution of the output from the convolution between a 3×3 filter. Nr.2 corresponds to the output of the same image, convolved with a 5x5 filter. Nr 3 was optioned with a 7×7 and Nr.4 with a 9×9 filter. The filters were trained for 55 epochs. The first 4 images and the last four images were made with filters that were initialized as scaled versions of one another. This shows clearly how differentiated the distribution can be, as well as the fact that they are not Half-Gaussian, but Normal-Distributions. This makes sense, because our input is not all-positive and our filters are not absolute Gabor-filters. Like most CNNs in image recognition our filters converge to Gabor-like behavior. But our experiments in using absolute values were not successful. We tried:

1. Add a Non-linearization layer between the convolution and the MinMax-pooling, that created absolute values.

2. After every training, turn the filter to absolute values.

3. Making the filters positive, by adding the largest negative value to every element in the filter.

Sadly none of these methods had any positive effect. The last two approaches stopped the filters from learning altogether while the first approach lowered the top-1 classification rate in Cifar-10 to under 50% and we still did not get a half- Gaussian in the value distribution.
Because of these problems to connect our approach to the basic idea we decided to try different thresholds and observe their impact on the functionality of the network.

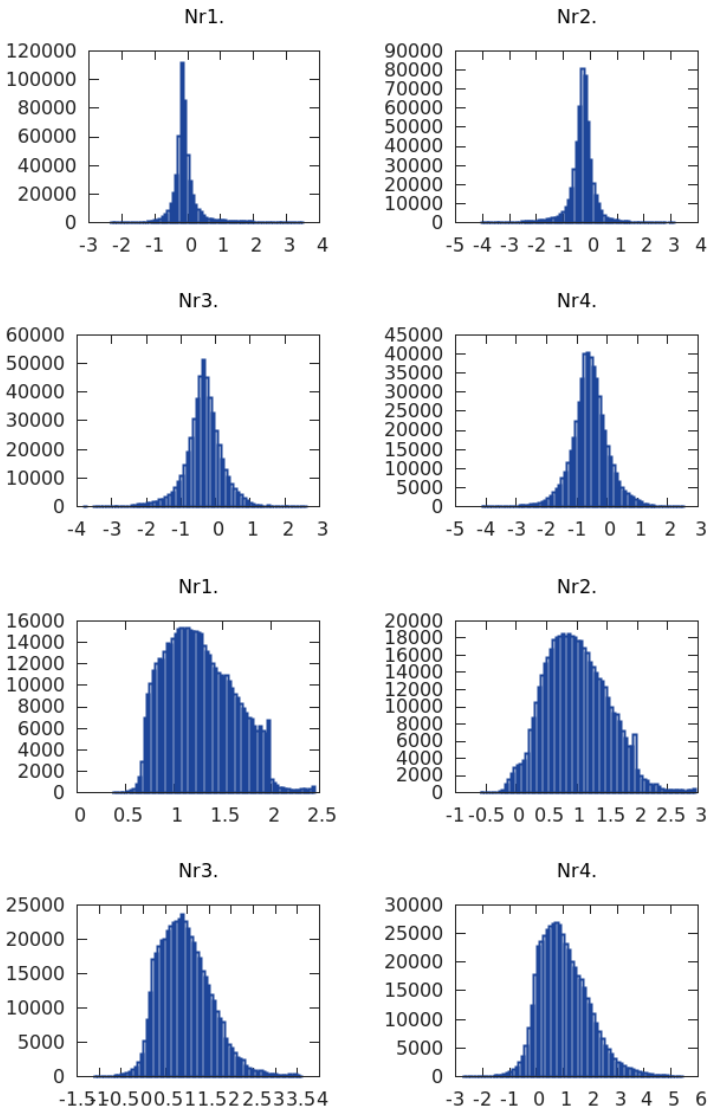


Figure 4.2: Histograms of the value distribution of the filter-layer output for our Cifar-10 MinMax implementation. Nr.1-4 show the results for scaled versions of the same filter. We can see that the four scales of the first filter follow normal distributions. The four scales of the second filter resemble broader normal distributions with wider variations and some out-laying values.

Chapter 5

Experiments

In order to test our implementation we compared three different implementations on two different datasets.

5.1 Dataset

At the beginning of the project we intended to use the ImageNet dataset, which is a large dataset with images of 256×256 pixels . But in the end we decided to go with two other datasets:

1. Cifar-10: Simple dataset with 10 Categories each containing 500 images of size 32×32 pixels.
2. MIT-Places: Complex dataset with 205 classes, 2.5 million total images of size 256×256 pixels.

We decided to go with these two datasets in place of ImageNet, because of two reasons. Firstly, we tested most of our programming on the smaller Cifar-10 dataset in order to save computational time.
Secondly we went with the combination of these two datasets as they should be able to highlight the advantages of our approach over the standard AlexNet. This is because the places dataset with its landscape shots has a higher variety of scale differences in the important parts of the image, compared to photographs of centered objects like the ImageNet database.

5.1.1 Cifar-10

Cifar-10 [13] is a relatively small dataset, both in terms of classes and the size of the images provided. It holds only 10 classes which represent animals or inanimate objects. The object to be recognized is mostly centered in the images. Examples of the images can be seen in figure 5.1.

This dataset has been used by a variety of image recognition publications [25, 12] even though its use has decreased over the last years. Since it

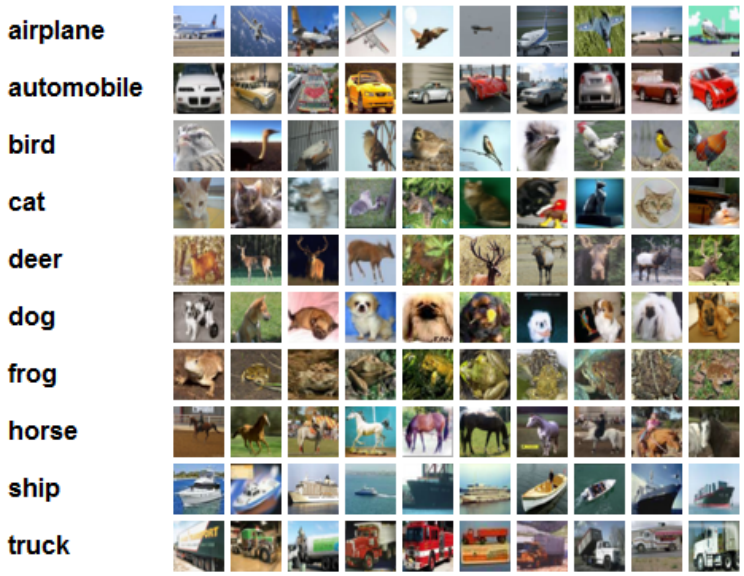


Figure 5.1: Image examples for all Cifar-10 categories. See https://www.cs.toronto.edu/~kriz/cifar.html

only has a small number of classes and the images themselves are only 32×32 pixels, it is not unusual for CNNs to perform with very high accuracy on Cifar-10 even though the same algorithms make far greater errors on larger datasets. Xu et al. [28] for example compared different activation functions on Cifar-10 as well as on cifar-100. Cifar-100 contains images of the same size, but provides 100 classes. Their results show, that the same algorithms that achieve around 99% accuracy on Cifar-10 were only able to reach around 60% on Cifar-100. This makes the Cifar-10 dataset a very situational database. On the one hand, we have to take results on it with a grain of salt. Judgment on the effectiveness of algorithms should not solemnly be made based on tests on this dataset. On the other hand, this can give a clear indication what results could be expected. As we could also see in the results of Xu et al., the tendencies that were observed in the Cifar-10 tests could also be found in the larger datasets. This makes Cifar-10 the ideal candidate for preliminary research. Especially since the dataset size is one of the major factors that determine how long an algorithm needs to be trained. For our tests we ran 55 epochs of our implementations on a PC with one Nvidia Titan-X. A test on Cifar-10 for most algorithms took between 3 and 8 hours. On the ImageNet and Places dataset however, the same implementations needed 2 to 5 days. Most preliminary studies would therefore be advised to first achieve results on Cifar-10 before porting the approach to more time consuming datasets.

Figure 5.2: Image examples for some MIT-Places categories. See http://blog.thehackerati.com/post/128746247346/deepdreaming-without-the-slugdogs

5.1.2 MIT-Places

The MIT-Places dataset [30] is an image dataset that consists of 205 categories. The categories have a wide range of places and scenes as can seen in figure 5.2. This dataset is considered to be a harder classification problem than datasets such as Cifar [13] or ImageNet. The reasons for this are twofold. Firstly it is a relatively big dataset, with a wide range of differences between the categories. Some categories, like different sorts of bedrooms are very close, while others like conference rooms and rain-forests are very different. This differentiates it from smaller datasets like Cifar-10 or even Cifar-100. But there are far bigger datasets, like ImageNet which holds 1000 categories. The big difference between these two datasets is the categories they depict. ImageNet is a database of objects and animals. Similar to Cifar most images in ImageNet have the object that needs to be identified somewhere in the image, mostly somewhere around the center. On the one hand this makes it easier to identify because all the reliable information can be found in one subpart of the image. On the other hand, this can also lead to problems. Some images depict more than one object. this can lead a classifier to be confused by the features extracted from non-relevant objects. Places is a different kind of dataset in this sense, because the classes are not bound to a specific object in the image. The classifier rather has to find the distinctive features in the image. This was already described by Zhou et al. [30] who showed that a CNN can be used to find the most informative regions to distinguish the images (see figure 5.3).

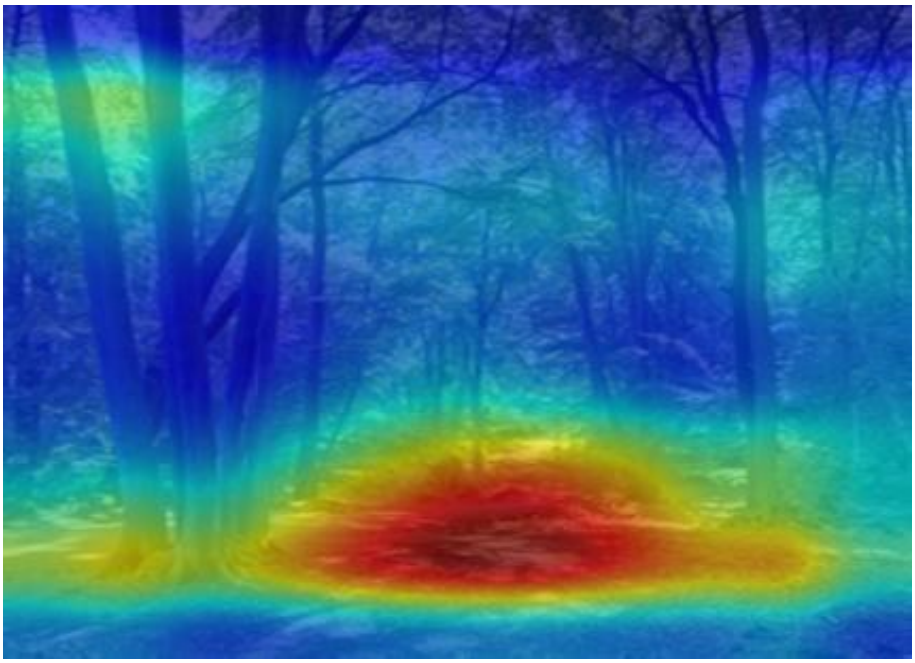


Figure 5.3: Image from The Places dataset, sowing the most informative region to identify the Image as a "forest path". See http://places.csail.mit.edu/demo/2.jpg

For this project we used a subset of the Places dataset consisting of 111 classes. The classes were randomly chosen, each containing 900 images. This was done in order to speed up the classification process and because we use an older, smaller type of network. Due to technical difficulties we had to train our networks on three different PCs and were not able to transfer the dataset every time. So we created new datasets on each PC with the same randomized algorithm and ran all networks on the datasets for the same number of times. For the PCs numbered (1), (2) and (3) this comes down to one run on (1), three runs on (2) and six runs on (3). This gave us a total of ten runs for each network type.
The PCs themselves also differ in configurations. PC (1) used a Nvidia Titan X graphics-card while PC (3) used a Nvidia Geforce GTX 1080. PC (2) was the Cartesius Supercomputer in Amsterdam. The specifications for which can be found here (https://userinfo.surfsara.nl/systems/cartesius/description).
This variation in PCs only applies for our training on the Places dataset. The Cifar-10 results were all obtained on PC (1).

5.2 Implementations

In order to test the effectiveness of our implementation we tested three different implementations on the datasets so we could compare the results.

1. AlexNet: The normal AlexNet implementation provided by Soumith
2. Max: Our implementation with parallel initialized filters, combined with Max-pooling
3. Min: Our implementation with parallel initialized filters, combined with MinMax-pooling

There are some consistencies over all our implementations, that should be mentioned here:
All of our convolution filters are randomly initialized. For all convolutions, the input is Zero-Padded, which means the input matrix is surrounded with fields containing value 0. The number of rows and columns added here is determined by the size of the filter. We used $(Filtersize-1)/2$ on each side. This ensures that the output of every convolution has the same size as the input.
All of our Max-pooling is done with kernel-size=2 and step-size=3.
Our fully connected layer uses Dropout with a chance of 50%.

5.2.1 AlexNet

In order to see if our method could increase the performance of a standard CNN architecture we used the implementation of AlexNet that is available at the github page of soumith[1], even though we made adjustments to the network when we applied it to the datasets. We did this because a bigger network would classify with such high accuracy, that differences between the different approaches would be hard to determine.

The specifications for AlexNet on Cifar-10 can be seen in Figure 5.4.

For the Cifar-10 dataset our network consists of two parallel streams. This two stream implementation is the standard implementation in the library for Torch, as it can make full use of Torch's ability to use two GPUs. This however was not important for us since we only used one GPU. Each stream begins with a convolution layer of 24 filters of size 5x5, followed by ELU-transformation and a Max-pooling layer. This pattern repeats two more times, with the slight change that both times the number of convolution filters is going up to 48. The third Max-pooling is then followed by uniting the two streams, by simply concatenating the output matrices. This gives a matrix of size $3 \times 3 \times 96$. The 3×3 is the result of the 36×36

[1] https://github.com/soumith/imagenet-multiGPU.torch

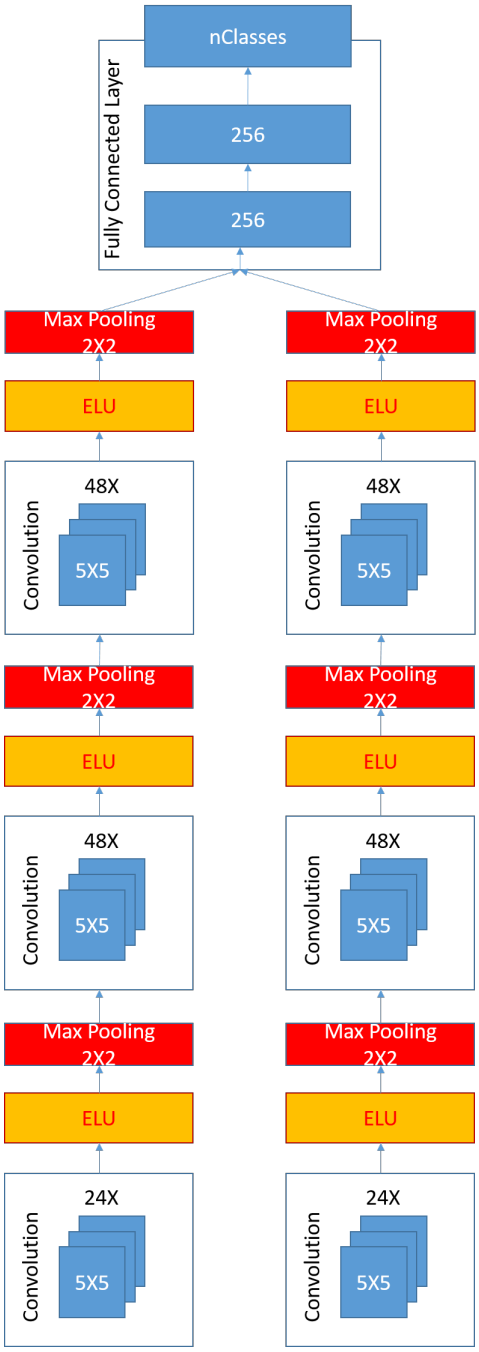


Figure 5.4: Structure of the AlexNet implementation used for the Cifar-10 dataset.

images passing through the three max-pooling layers. The matrices are concatenated on the third dimension. For the 48 filters of the last convolution layer that gives a size of 96. The fully connected layer for the classification has 256 units in the first two layers and 10 output units for the 10 classes, utilising the softmax function.

For the Places dataset the net is deeper, and the fully connected layers have more units. This can be seen in figure 5.5.

Here we also have two streams, but each stream has five convolution steps instead of three. The first two filters are still followed by both ELU and Max-pooling, but the two filters after that only have the non-linearization and no pooling, until the last filter is again followed by both. We also do not have 5×5 filters at every stage but 24 times 11×11 filters in the first layer, followed by 48 5×5 filters in the second and 24 3×3, 24 3×3 and 48 3×3 in the last layers.
The fully connected layer in this case has 4069 units in the hidden layers, followed by 111 output units.
The parallel architecture differs from our max and min implementation but after comparing a parallel and a non-parallel implementation of AlexNet we could not find a difference in performance.

5.2.2 Max

The structure of both the Max and the MinMax implementation are the same. The only difference between the two is how the dimensional pooling is done after the first convolution. The structure for both on the Cifar-10 can be seen in figure 5.6 and the structure for the places dataset in 5.7.

This implementation is only single-stream, in contrast to the normal two-stream implementation. A two stream implementation would be advisable if one has the possibility to run the network on two GPUs, but since we only had one GPU to our disposal the one stream alternative did not make any difference.
The first layer has four different sized convolution parts. Every cluster has 12 filters of sizes: 9×9, 7×7, 5×5 and 3×3. These Filters are initialized differently from the other convolutions layers, since only the 9×9 filters are initialized with random values, while the other filters consist of scaled versions of it. The calculation and back-propagation however is done the same as every other convolution.
The output of the four convolutions in combined in the Dimensional Max pooling, where the four $36 \times 36 \times 12$ matrices are combined into one $36 \times 36 \times 12$ matrix.
This matrix is then passed through the ELU and Max pooling layers. The same as in the basic network, this is followed by two more iterations of the convolution/ELU/Pooling layers, with the difference that in this implemen-

Figure 5.5: Structure of the AlexNet implementation used for the Places dataset.

Figure 5.6: Structure of the Max/MinMax implementation used for the Cifar-10 dataset.

tation the first convolution is done with 96 5×5 filters while the second consists of 96 3×3 filters.
The fully connected layer is made out of two hidden layers with 256 units, followed by the output layer of 10 units.

5.2.3 Min

The structure of both the Max and the MinMax implementation is the same. The only difference between the two is how the dimensional pooling is done after the first convolution. The structure for both on the Cifar-10 can be seen in figure 5.6and the structure for the places dataset in 5.7. For a detailed description of the Network structure see section 5.2.2.

5.2.4 Threshold

As described in section 4.3.2 the theory behind our implementation had some problems around the thresholds for the MinMax function. So we decided to explore multiple options to see how they would impact the classification behavior of the network. Since the output of our first convolution layer follows a normal distribution, as shown in figure 4.2, we compare three different implementations of the threshold.

1. one-tailed threshold (Figure 5.8)
2. two-tailed threshold (Figure 5.9)
3. fixed threshold

For the one tailed threshold we tried an upper bound as well as a lower-bound with multiple different thresholds that were dependent on the mean and standard deviation. We tested if the value was above ($>$) or below ($<$) a certain threshold-value. That threshold value is computed by using functions of mean and standard deviation (std) of the probability distribution. In total we tested:

1. $value > mean + std$
2. $value < mean - std$
3. $value > mean + 2 * std$
4. $value < mean - 2 * std$
5. $value > mean$
6. $value < mean$
7. $value < mean + std$

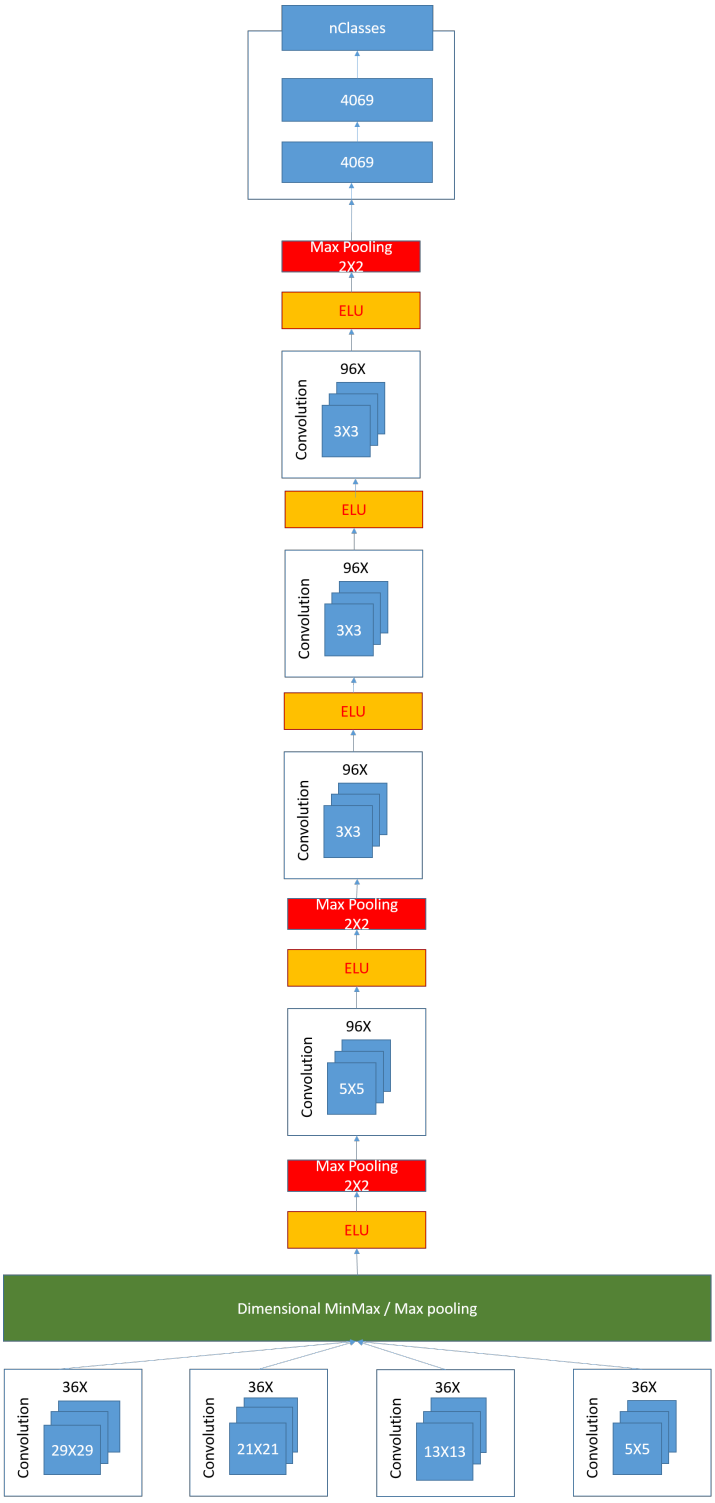


Figure 5.7: Structure of the Max/MinMax implementation used for the Places dataset.

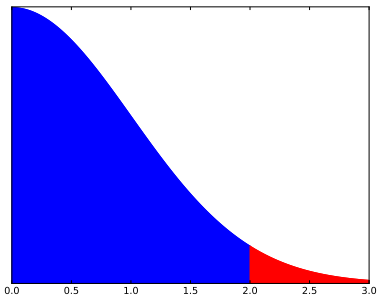


Figure 5.8: One Tailed distribution

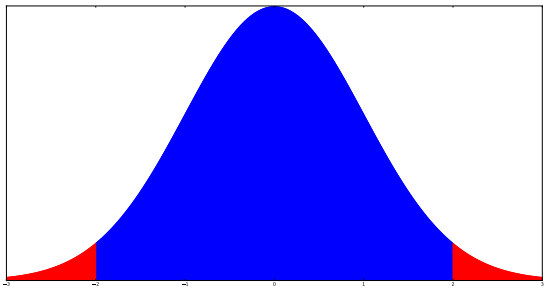


Figure 5.9: Two Tailed distribution

8. $value > mean - std$

From all one sided thresholds the best average score was obtained by item 1. Items 2, 3, 4, 5 and 6 performed nearly as good as item 1, while 7 and 8 were significantly worse.
For the two tailed threshold we also tested different variations of changing thresholds depending on mean and standard deviation. We tested:

1. $(value < mean - 2 * std) \vee (value > mean + 2 * std)$
2. $(value > mean - 2 * std) \vee (value < mean + 2 * std)$
3. $(value > mean - std) \vee (value < mean + std)$
4. $(value < mean - std) \vee (value > mean + std)$

The highest results were obtained by item 1 even though none of the others performed significantly worse.

Since the differences between the thresholds were so unremarkable we also decided to test for a fixed threshold. For this we simply tried:

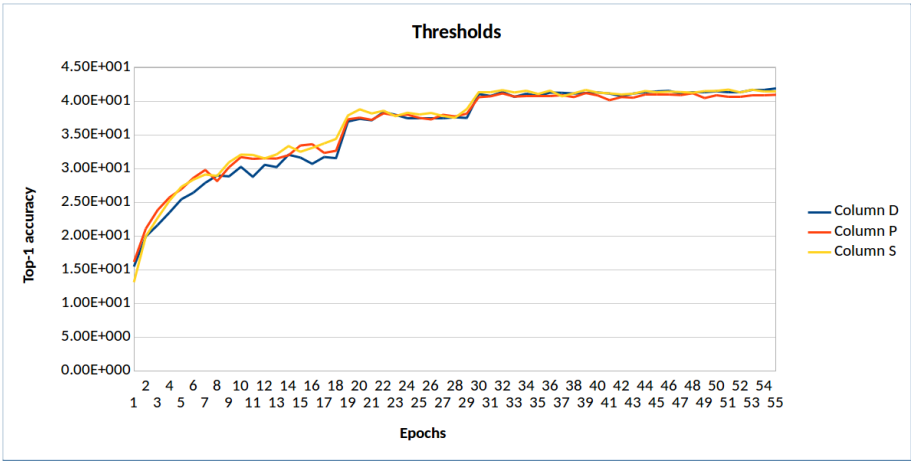


Figure 5.10: Results for the best thresholds of each category. Mean_2std is the two-tailed threshold for $(value < mean - 2 * std) \vee (value > mean + 2 * std)$. mean_std is the one-tail threshold of $value > mean + std$. fixed_0 is the fixed threshold of $value > 0$. These results were obtained on the places dataset.

1. $value > 0$
2. $value < 0$

There was no difference in the obtained results from either of those possibilities.
In order to determine the best possible solution for our network, we then compared the best solutions from the three approaches. The results, which can be seen in figure 5.10, show that there is no significant difference between the three approaches after around 20 epochs. The two-sided threshold appears to perform worse in early stages of the training, but it cannot be ruled out that this is a result of an unfortunate random initialization of the filters. Overall the results indicate that elaborate threshold functions in the MinMax algorithm do not appear to have any positive impact on the classification results.

The results shown here were obtained with our MinMax implementation on the Places dataset. We performed early testing on the Cifar-10 dataset, but we could not exclude the possibility that the lack of differentiation between the approaches was a result of the simplicity of the data. Therefore it seemed necessary to repeat the testing on the Places dataset, which confirmed the results.
We would also like to note, that this does not mean that the threshold itself is irrelevant. The threshold value itself might not be important, but since the filters that produce the values which are passed through the MinMax function are still learned, the threshold value does not need to change in or-

der for the network to adapt to different noise levels. Since the filters are not co-dependent, and trained separately, the network has the opportunity to adjust the values for different sized filters in order to optimize the resulting convolution for different noise levels.

5.3 Hypotheses

Due to the differences in the datasets we expect our algorithm to outperform both the Max implementation as well as the basic AlexNet on the Places dataset. For the Cifar-10 we expect our network to perform closer to the normal AlexNet and possibly be outperformed by the max implementation. This is due to the fact that our implementation should have an advantage when different scales of filters can occur over the images. Since the Cifar-10 dataset, as described in section 5.1.1 has images of only 32×32 pixels, there should not be much room for different scales in the edges. The places dataset however has images of big sceneries, where the distance should provide different scaled edges, especially since the images are more detailed with 256×256 pixels.

Chapter 6

Results

Since this project concerned itself with both the effectiveness of our approach and the behavior of our new model, we will report on the performance of our networks, as well as the training of the filters.

6.1 Networks

As described above, we have tested different configurations of our network, in order to compare them with one another and with the basic AlexNet as well as a Max implementation. Our measure of performance for the networks is the top-1 accuracy. This means that we compare how often the networks have the correct class as their first guess. We do this despite the fact that many related articles use the top-5 accuracy instead. We do this because top-5 accuracy compares the number, where the correct class is in the top-5 predictions. As the Cifar-10 dataset only contains ten classes the top-5 accuracy usually scores very high for most networks. Since we wanted to see the differences between the networks clearly and did not want different measures for the two datasets we went with the top-1 accuracy.
In order to test different thresholds for the MinMax implementation, we tried different configurations which were described, with their results, in section 5.2.4. For the results described in this segment we used the implementation of MinMax, that utilized the two-tailed threshold $(value < mean - 2 * std) \vee (value > mean + 2 * std)$ as the results were highest amongst all thresholds, even though the differences were not significant.
All results reported were obtained from a test-set that had no overlap with the training-set.
We first implemented and tested our three networks on the Cifar-10 dataset.

As we can see in table 6.1 and figure 6.1 on Cifar-10 the results of all three methods are very close. In the beginning, the Min implementation performs worst, but overtakes AlexNet and Max consistently after 24 epochs. After all 25 training epochs the MinMax approach performs best with 82.1%

Implementation	Average Top-1 accuracy	Variance
AlexNet	81.5%	10.62
Max	81.3%	2.55
MinMax	82.1%	0.733

Table 6.1: Results for Top-1 accuracy and the Variance of the accuracy for the three implementations on the Cifar-10 dataset. Averaged over 55 runs

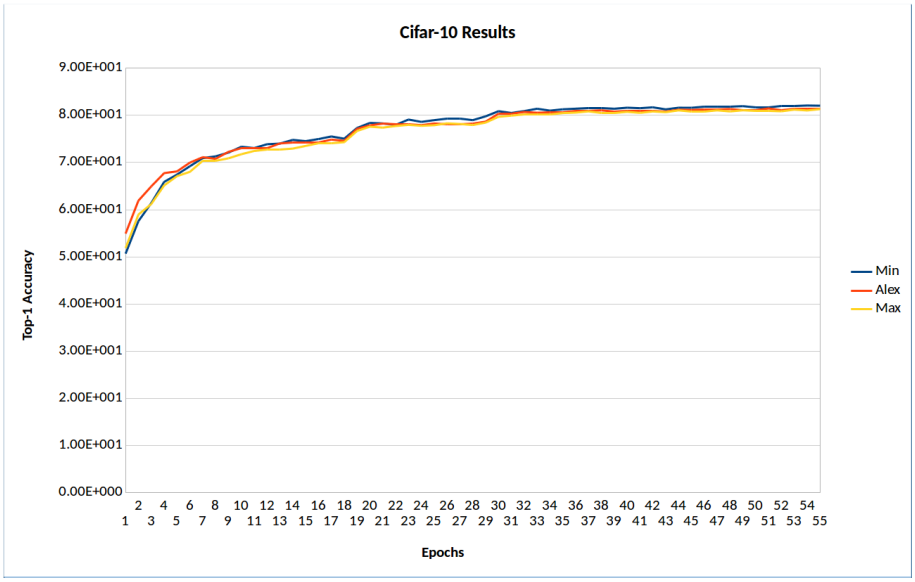


Figure 6.1: Results of the AlexNet, Max and MinMax implementation on the Cifar-10 dataset. Given in Percentage top-1 accuracy. Averaged over 10 Runs.

Implementation	Average Top-1 accuracy	Variance
AlexNet	38.69%	0.23
Max	36.19%	15.69
MinMax	38.90%	6.98

Table 6.2: Results for Top-1 accuracy and the Variance of the accuracy for the three implementations on the Places dataset. Averaged over 55 runs.

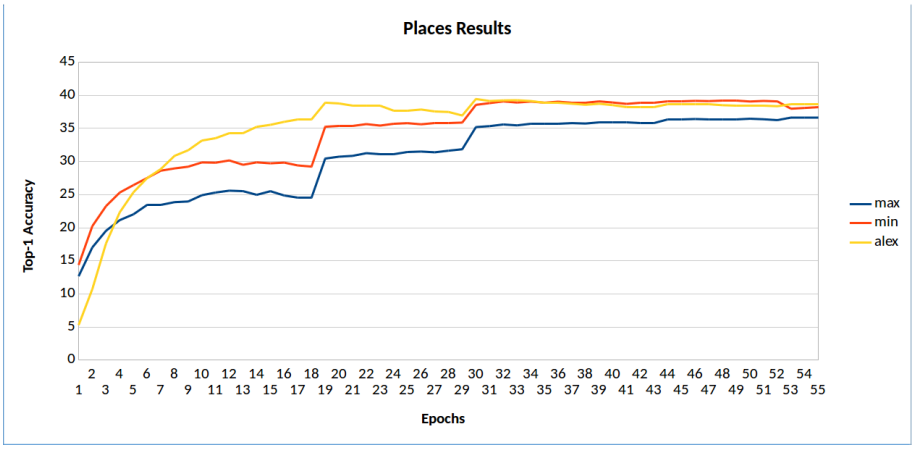


Figure 6.2: Results of the AlexNet, Max and MinMax implementation on the Places dataset, averaged over all ten runs. Given in Percentage top-1 accuracy.

Top-1 recognition rate but the differences to neither Max (P value: 0.18) nor AlexNet (P value: 0.53) are statistically significant. Between AlexNet and Max, AlexNet performed better with an averaged 81.5% to 81.3% recognition rate but the difference was also not found to be significant (P value: 0.88).

The results on the Places dataset can be seen in table 6.2 and figure 6.2. These results were obtained by averaging all the available data, which are ten runs on our Places datasets. Due to technical difficulties we had to run on different PCs with different subsets of the dataset. Specifics over these runs can be found in section 5.1.2.
The results show, that after 55 epochs MinMax performs best with 38.9% recognition rate, followed by AlexNet with 38.69% and lastly Max with 36.19%. Interestingly the order of variance in the networks is the other way around, with AlexNet having the smallest variance of 0.23 followed by MinMax with 6.98 and finally Max with 15.69. However regarded with a two-tailed P-test none of the differences are statistically significant, even though Max and AlexNet are near the 5% certainty (P-Value 0.057). For details see table 6.3.

P-Values	AlexNet	Max	MinMax
AlexNet	–	0.057	0.79
Max	0.057	–	0.084
MinMax	0.79	0.084	–

Table 6.3: P-Values between the Networks on the Places dataset for a two-tailed test.

Implementation	Average Top-1 accuracy	Variance
AlexNet	39.3%	0.03
Max	40.87%	0.02
MinMax	41.2%	0.05

Table 6.4: Results for Top-1 accuracy and the Variance of the accuracy for the three implementations on the Cifar-10 dataset. Averaged over the three runs with the best performance.

We also decided to compare the best three runs for all three networks. This gave us the results seen in table 6.4 and graph 6.3.

Here we can see that MinMax has no drop-off in the last epochs. This makes it significantly better than AlexNet (P-Value 0.01). Even Max performs significantly better than AlexNet here (P-Value 0.013). The differance between MinMax and Max however is still not significant (P-Value 0.127). Lastly we took a look at epoch 51, which is the last epoch before the drop-off in MinMax. Here MinMax (10 point average 40.36) performs higher than Max (10 point average 35.97) which is statistically significant (P-Value 0.038). On the other-hand the difference to AlexNet (10 point average 38.49) is not significant (P-Value 0.1)

Figure 6.4 shows a comparison between the Cifar-10 and Places results. We can see that the differences on the Places dataset are a lot less distinct than the results on Cifar-10 but in order to analyse the difference we need to look at the perceptual improvement of MinMax over the other two algorithms. After the overfitting MinMax is only a 0.54% improvement on Places while it improved on Cifar-10 with 0.71%. But if we consider the top-3 networks the improvement grows to 4.9% and if we look at the improvement at the 51st epoch, before the drop-off in MinMax, we also see an improvement of 4.9%.

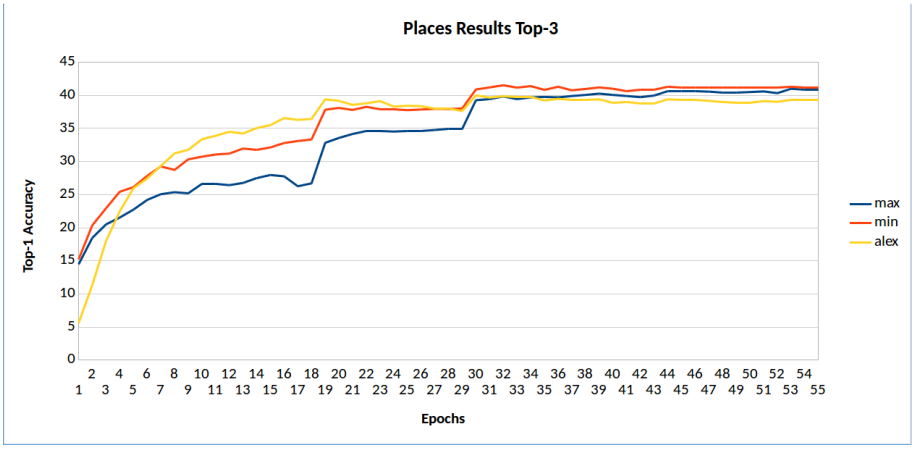


Figure 6.3: Results of the AlexNet, Max and MinMax implementation on the Places dataset, averaged over the best three runs of the total ten runs. Given in Percentage top-1 accuracy.

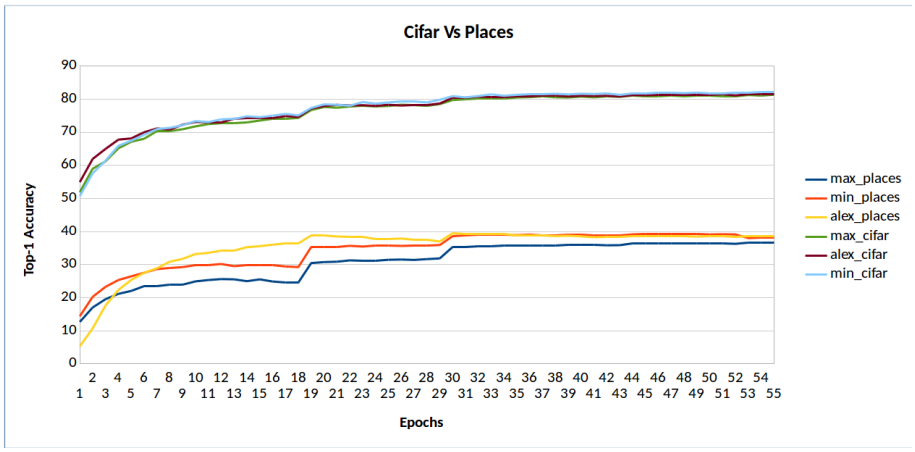
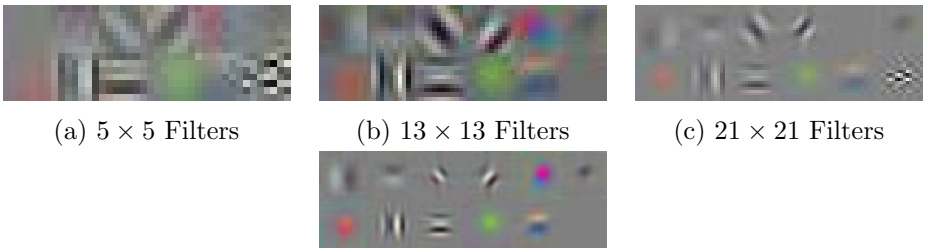


Figure 6.4: Comparison of the implementations on the Places and Cifar-10 dataset. Given in Percentage top-1 accuracy.

6.2 Filters

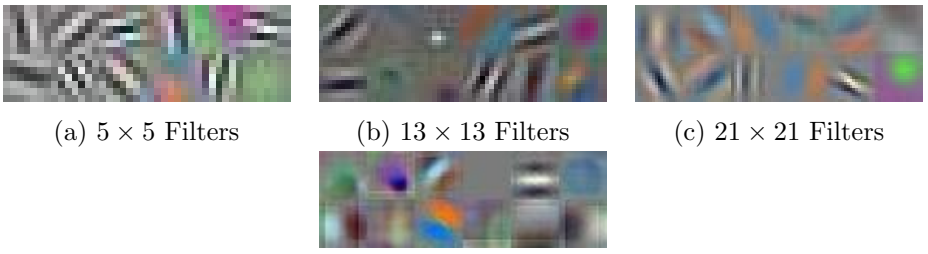
The filters of the first layer are an important part of this project. As mentioned before, our implementation does not train the filters dependently. but we were still interested to see if some form of behavior would emerge, that would facilitate filters in the same position to detect similar shapes in different sizes. To reiterate our implementation; the filters in the same position over different sizes, compete to have their value passed on to the next layer for every pixel. This is why it is also referred to as "competitive pooling" [16]. During the following back propagation only the filters of the winning values are updated. This means, that even though the filters are updated independently, they are still connected via the MinMax/Max module.
Our hope was that even though we did not explicitly train the filters to be scaled versions of one another they would still assume similar shapes in different scales. This would give us an indication, that the brain could also be using a similar method, that results in neurons reacting to different sized features, and does not need to have a mechanism that forces such a structure on them. For this we wanted to know if such similarity happens in our MinMax filter, and if so, are they more prominent than in other "competitive pooling algorithms", specifically our Max implementation.
In order to test this we did two things. First we looked at visualized versions of our first level filters to see if such a pattern could be visible. Examples of this can be seen in Figure 6.5 and Figure 6.6. Figure 6.5 shows the Filters that we got in a trained MinMax network. Figure 6.6 shows the filters from a trained network that used Max instead. The filters in the same positions correspond to the filters that were competing via MinMax/Max. Every image ((a),(b),(c) and (d)) correspond to one filter size. Each of those depicts all 12 Filters of that size. An example of this can be seen in the bottom right corner of all the images. In the bottom right off Figure 6.5(a) and (c) we can see checker pattern, while the same space in (b) and (d) shows just grey space. Those four squares are four filters of different sizes, that compete for producing the output. There are several things that we can see in these images. Firstly the different sizes of the filters share more similarities in Figure 6.5 than in Figure 6.6. There are multiple instances where we can see the same forms, shapes and colors in the same position over all or multiple images in Figure 6.5, whereas the only example in Figure 6.6 that could be considered an approximation is the bottom left image in (a) and (b) and the second from the right in the top row, in (b) and (d). This seems to be a clear indication that the filters in MinMax turn our more correlated than the filters in Max. But it definitely should be mentioned that we chose the best example for the MinMax filters that we could find. Other filters generated by MinMax were not this clear, but we could not find a single example in the Max filters that would come even close.
It also stands to note that we did not definitely say, that they are all scaled



(a) 5×5 Filters (b) 13×13 Filters (c) 21×21 Filters

(d) 29×29 Filters

Figure 6.5: Trained MinMax filters on Places dataset after 55 epochs



(a) 5×5 Filters (b) 13×13 Filters (c) 21×21 Filters

(d) 29×29 Filters

Figure 6.6: Trained Max filters on Places dataset after 55 epochs

versions of one another. An example of this can be seen in the third filter from the right in the bottom row. This filter appears to be a green field in (a) and (b) and is reduced to a green dot in (c) and (d). Since the filters get increasingly bigger, the field covered by green by (c) or (d) is approximately the same size as (b). But since they are competing with one another they cannot react to the exact same input, so the slight differences have to be significant for the algorithm.
In other cases, like the bottom right of Figure 6.5, some of the filters show similar trained shapes ((a),(c)) while the others are not trained ((b),(d)). This could be a result of those two instances covering the entire range of this feature in the dataset.
Another interesting case is depicted in the third filter from the right in the top row of figure 6.5. Here the pattern depicted in (a) and (d) is the inverse of the pattern from (b) and (c).
All of this seems to indicate that a MinMax algorithm puts an indirect pressure on the competing filters into some kind of not yet discovered relation.

The second test we performed, was to calculate the structural similarity (SSIM) between the filters. The SSIM index was invented by Wang et al. [26] in order to assess image quality by comparing images with corrupted variations. It compares images with sliding windows. For two windows a and b, that have the same size and are in the same position on the two images to compare, the SSIM value is:

	SSIM Mean	SSIM STD
MinMax	0.7588	0.2006
Max	0.686	0.249
Random Filters	-0.0011	0.1002
Random Picked Filters	0.392	0.324

Table 6.5: Averaged structural similarity values and standard deviation for the different filtering methods.

$SSIM(a,b) = \frac{(2\mu_a\mu_b+c1)(2\sigma_{ab}+c_2)}{(\mu_a^2+\mu_b^2+c_1)(\sigma_a^2+\sigma_b^2+c_2)}$

Here μ stands for the average of the window, σ is the windows variance and c are variables for stabilization. a and b are the respective windows that are compared. Taking the mean of all the window comparisons gives the SSIM value for the two images.
For our comparison we scaled all the filters of our networks to 13x13 and then performed the SSIM between the filters at the same positions and we did this for every filter-size with every filter-size. We used windows of size 7×7 for the SSIM. We used 13×13 for the filters because we wanted to use downscaling as much as possible, but did not want to choose the 5x5 filters because we were worried to lose too much information on this scale. We chose 7×7 for the windows because it gave good results.
We then compared the results on this for MinMax, Max, randomly initialized filters and four randomly picked filters from different trained networks. If our filters are indeed influencing their competing counterparts on the different scales, we would expect to see a significantly higher structural similarity between the MinMax filters than the max, and the same between MinMax/Max and Random-Filters/Randomly-Picked-Filters.
The results can be seen in Table 6.5.

The random filters and randomly picked filters serve as our baseline. The random filters show the value for absolutely nothing comparable in the image. Every similarity found is absolutely random. The randomly picked filters are the baseline for the structural similarities that cannot be attributed to any type of indirect influence that a network can have, since they are randomly picked from different networks. So their values come from the existence of filters, but that are not the same.
The results in table 6.5 show exactly what we would expect. The randomly picked filters are more similar than the random-value filters. While the filters in Minmax and Max that use competitive pooling have a higher similarity than both. The difference between MinMax and Max is small but still significant (P-value 0.034). This means that the competing filters in MinMax are the most similar.

Chapter 7

Conclusion

Our results indicate that MinMax pooling improves the performance of networks, however we need to take a closer look at the circumstances. The results show only a significant improvement on Places for our algorithm, before the 52nd epoch or for a comparison of the top three networks. What we need to note about this is that all top three networks, except for one network for AlexNet, were made on one subset of the places dataset. As we mentioned in subsection 5.1.2 we used three different places subsets because of problems we had with our PCs. All these subsets were created with the same randomization process, but by chance the subset on which we performed three runs of each network, appears to be more favorable than the other two. However, the improvement for the MinMax and Max algorithms is a lot bigger than for AlexNet that improved barely noticeable. From what we have seen in the filters, this could mean that the dataset on which MinMax performed better, had a higher amount of scale differences for the filters to learn, which would favor the MinMax algorithm most and the AlexNet least, if at all. Another explanation could be that the dataset on which we performed the six runs for each network had a very small amount of scale differences. This could explain why the MinMax algorithm might overfit at the end, leading to the performance drop-off after epoch 51 (figure 6.2). If this is the case, we might need to be careful what configuration of networks can be used on specific datasets and it should definitely be investigated what exact criteria can lead to these drastic changes.
If we look at the percentage improvement we get the same picture as for the overall performance. On Cifar-10 MinMax improved over AlexNet by 0.71% while on the places dataset overall the improvement was only 0.54%. But if we only look at the first 51 epochs of Places, MinMax improved AlexNet by 4.9% which means that the improvement is nine fold for the more complex dataset.
In order to put our results in perspective we would also like to look at the results obtained by the Max algorithm. It came as a surprise to us that the

Max algorithm performed worse than AlexNet, especially since Liao et al. [16] had shown that it can improve more complex networks. But there are two things that we need to note about the differences between our and their Max implementation. Their Max uses Max repetitively in every layer for a very deep network structure. We only use the Max once, in our first layer. This could mean that the benefit of Max does not lay in the scale selection property but a different mechanism that is either useful in deeper layers or because of repetition. They also used the Cifar-100 and SVHN dataset, both with smaller 32x32 images that contain very little visual information for scale variance.
Overall it can be said, that the results show the direction we were expecting, even though the differences were not as clear as we hoped, due to the over fitting of our network. We can definitely see a promise for these kinds of scale sensitive modules and hope that future research can extend upon our experiments.

7.1 Future Work

This work is a pilot study for the basic properties of multi-scale filters with MinMax pooling in CNNs, but there are a lot of questions in this area that still need to be answered. Most pressing of course is to address the problem of over-fitting, that the MinMax networks exhibit on some of the more complex datasets. It would be very good to find out what type of datasets facilitate this behavior and what impact it has on the learned filters. We briefly looked into the differences between the filters of the first layer, before and after the over fitting, but could not find any obvious differences.
Since this idea came from the models we have of the workings of the human brain, we need to answer the question, if this implementation actually performs more human like, than standard implementations. To this end we have to compare the mistakes our network makes on a dataset with the mistakes made by the standard approaches and the mistakes of human beings. If our implementation is indeed closer to the workings of the human brain, the errors made should also correspond closer to human performance than the normal networks.
It should also be tested how the performance of networks changes when the MinMax module is applied to deeper parts of the networks. This could very nicely be tested and compared with the implementations of Liao et al. [16] since they already compare the deeper multi-scale inception network with deep-layer max-pooling. This implementation seems optimal to compare to our MinMax approach. Answers in this direction could lead us to hints, as to why the brain does not need to rely on deeper networks and could help us to break away from the current trend to just extend networks deeper and deeper to classify more complex data. We also should look further into what

we know about the workings of the brain and continue to try to impose this on our network structures. As this thesis showed, this approach can yield interesting results that might improve the workings of neural networks even further. It could be especially promising to look into things such as the parallel working streams of the brain, or even go away from approaches that only use spatial relations and look into ideas that also incorporate a time component. This could either be for data that have a time component, as is currently done with Recurrent Convolutional Neural Networks [18], but also the newer Spiking Networks [3], that are very close to what we know about the human brain.

Bibliography

[1] Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, and Gerald Penn. Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition. In *2012 Institute of Electrical and Electronics Engineers (IEEE) international conference on Acoustics, speech and signal processing (ICASSP)*, pages 4277–4280, 2012.

[2] Yoshua Bengio. Learning deep architectures for AI. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[3] Sander M Bohte, Joost N Kok, and Han La Poutre. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, 48(1):17–37, 2002.

[4] Peiqiu Chen, Hanli Wang, and Jun Wu. Correlative filters for convolutional neural networks. In *Systems, Man, and Cybernetics (SMC), Institute of Electrical and Electronics Engineers (IEEE) International Conference on*, pages 3042–3047, 2015.

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition. (CVPR) Institute of Electrical and Electronics Engineers (IEEE) Conference on*, pages 248–255, 2009.

[6] James H Elder and Steven W Zucker. Local scale control for edge detection and blur estimation. *Institute of Electrical and Electronics Engineers (IEEE) Transactions on pattern analysis and machine intelligence*, 20(7):699–716, 1998.

[7] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[8] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, and Tara N Sainath. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Institute of Electrical and Electronics Engineers (IEEE) Signal Processing Magazine*, 29(6):82–97, 2012.

[9] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[10] David H Hubel and Torsten N Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243, 1968.

[11] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the Institute of Electrical and Electronics Engineers (IEEE) conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[12] Alex Krizhevsky and G Hinton. Convolutional deep belief networks on Cifar-10. *Unpublished manuscript*, 40, 2010.

[13] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[15] John Le Cun, B Boser, John S Denker, D Henderson, Richard E Howard, W Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, 1990.

[16] Zhibin Liao and Gustavo Carneiro. Competitive multi-scale convolution. *arXiv preprint arXiv:1511.05635*, 2015.

[17] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *2015 Institute of Electrical and Electronics Engineers (IEEE) Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 427–436, 2015.

[18] Pedro HO Pinheiro and Ronan Collobert. Recurrent convolutional neural networks for scene labeling. In *International Conference on Machine Learning (ICML)*, pages 82–90, 2014.

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

[20] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[21] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[22] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the Institute of Electrical and Electronics Engineers (IEEE) Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[23] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[24] Jos van de Wolfshaar, Mahir F Karaaba, and Marco A Wiering. Deep convolutional neural networks and support vector machines for gender recognition. In *Computational Intelligence, 2015 Institute of Electrical and Electronics Engineers (IEEE) Symposium Series on*, pages 188–195, 2015.

[25] Shengjie Wang, Abdel-rahman Mohamed, Rich Caruana, Jeff Bilmes, Matthai Plilipose, Matthew Richardson, Krzysztof Geras, Gregor Urban, and Ozlem Aslan. Analysis of deep neural networks with the extended data jacobian matrix. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 718–726, 2016.

[26] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *Institute of Electrical and Electronics Engineers (IEEE) transactions on image processing*, 13(4):600–612, 2004.

[27] Andrew Witkin. Scale-space filtering: A new approach to multi-scale description. In *Acoustics, Speech, and Signal Processing, Institute of Electrical and Electronics Engineers (IEEE) International Conference on ICASSP'84.*, volume 9, pages 150–153, 1984.

[28] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[29] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833. Springer, 2014.

[30] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using Places database. In *Advances in neural information processing systems*, pages 487–495, 2014.