



**CHALMERS**  
UNIVERSITY OF TECHNOLOGY

---

# **Deep learning methods for recognizing signs/objects in road traffic**

Bachelor's thesis in Electrical Engineering, EENX15

Josef Bengtson, Filip Heikkilä, Per Nilsson, Lukas Nyström,  
Erik Persson and Gustav Tellwe

Supervisor: Ebrahim Balouji, Electrical Engineering



BACHELOR'S THESIS 2018

# Deep learning methods for recognizing signs/objects in road traffic

Josef Bengtson, Filip Heikkilä, Per Nilsson, Lukas Nyström, Erik  
Persson, Gustav Tellwe



Department of Electrical Engineering  
Signal processing  
CHALMERS UNIVERSITY OF TECHNOLOGY  
Gothenburg, Sweden 2018

Deep learning methods for recognizing signs/objects in road traffic.  
Josef Bengtson, Filip Heikkilä, Per Nilsson, Lukas Nyström, Erik Persson, Gustav  
Tellwe

Supervisor: Ebrahim Balouji, Electrical Engineering  
Examiner: Irene Yu-Hua Gu, Electrical Engineering

Bachelor's Thesis 2018  
Department of Electrical Engineering

Chalmers University of Technology  
SE-412 96 Gothenburg

Typeset in L<sup>A</sup>T<sub>E</sub>X  
Gothenburg, Sweden 2018



Deep learning methods for recognizing signs/objects in road traffic.  
Josef Bengtsson, Filip Heikkilä, Per Nilsson, Lukas Nyström, Erik Persson, Gustav Tellwe ©  
Department of Electrical Engineering  
Chalmers University of Technology

## Abstract

In this thesis the authors have used three different neural network structures, Convolutional Neural Network (CNN), Capsule Network and Faster R-CNN, to detect and classify traffic signs from the German Traffic Sign Recognition- and Detection Benchmark (abbreviated GTSRB and GTSDb). The networks were implemented using machine learning libraries for Python such as TensorFlow and Keras. The results obtained were a classification accuracy of 98.61 % for the CNN and 99.62 % for the Capsule Network. The aim was to attain results close to the results of the best network listed at GTSRB, which is 99.71%. Another goal was to use a deep learning networks to locate and classify a traffic sign within an image. This was done with an implementation of a Faster R-CNN, which was able to successfully detect and classify signs with an mAP of 0.56. It was noted that the network was able to detect most of the signs, however it struggled with classification. For this reason a version that combined Faster R-CNN with a Convolutional Neural Network trained on GTSRB was created, which outperformed the Faster R-CNN.

Keywords: CNN, Capsule Network, Faster R-CNN, GTSRB, GTSDb, image recognition, Artificial intelligence, deep learning

## Sammandrag

I den här uppsatsen har författarna använt tre olika neurala nätverksstrukturer, Convolutional Neural Network (CNN), Capsule Network och Faster R-CNN, för att hitta och klassificera trafikskyltar från German Traffic Sign Recognition- and Detection Benchmark (förkortat GTSRB och GTSDb). Nätverken implementerades med maskininlärningsbibliotek för Python, som TensorFlow och Keras. Resultaten var en klassificeringsprecision på 98.61 % för CNN och 99.62 % för Capsule Network. Målet var att få ett resultat nära resultatet av det bästa listade nätverket på GTSRB, vilket är 99.71 %. Ett annat mål var att använda djup maskininläring för att lokalisera och klassificera skyltar i en bild. Det gjordes med en implementation av Faster R-CNN som klarade av att framgångsrikt hitta och klassificera skylter med mAP på 0.56. Det noterades att nätverket klarade av att lokalisera de flesta skyltarna, men hade problem med klassificeringen. Av den anledningen skapades en variant som kombinerade Faster R-CNN med ett CNN tränat på GTSRB, som överträffade Faster R-CNN

Nyckelord: CNN, Capsule Network, Faster R-CNN, GTSRB, GTSDb, bildigenkänning, artificiell intelligens, djup maskininläring

## Acknowledgements

We would like to express our deepest appreciation to our supervisor Ebrahim Balouji who has helped us throughout the project.

We are also grateful for the assistance given by Michael Oberparleiter and the department of Space, Earth and Environment in offering computational resources.

The authors, *Gothenburg May 2018*

## Abbreviations

**ANN** Artificial Neural Network

**CNN** Convolutional Neural Network

**conv layer** convolutional layer

**GTSDB** German Traffic Sign Detection Benchmark

**GTSRB** German Traffic Sign Recognition Benchmark

**mAP** Mean Average Precision

**MNIST** Modified National Institute of Standards and Technology

**R-CNN** Regions with CNN features

**SGD** Stochastic Gradient Descent

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Aim . . . . .	2
1.2	Constraints . . . . .	2
1.3	Social and ethical aspects . . . . .	2
1.4	Disposition . . . . .	3
<b>2</b>	<b>Theory</b>	<b>4</b>
2.1	Artificial neural networks . . . . .	4
2.1.1	Neurons . . . . .	4
2.1.2	Network structure . . . . .	5
2.1.3	Supervised learning . . . . .	6
2.1.4	Cost function . . . . .	6
2.1.5	Optimizers . . . . .	7
2.1.6	Backpropagation . . . . .	8
2.2	Regularization . . . . .	9
2.2.1	Dropout . . . . .	10
2.2.2	Batch normalization . . . . .	11
2.3	Image preprocessing and data augmentation . . . . .	11
2.4	Convolutional Neural Networks . . . . .	11
2.4.1	Convolutional layers . . . . .	12
2.4.2	Max pooling . . . . .	13
2.4.3	Fully connected layers . . . . .	13
2.4.4	Capsule Network . . . . .	14
2.5	Networks for object detection . . . . .	16
2.6	Evaluating the results . . . . .	18
2.6.1	Precision, Recall and $F_1$ -score . . . . .	18
2.6.2	Mean average precision . . . . .	19
2.6.3	Confusion matrix . . . . .	19
2.6.4	Class clustering . . . . .	20
<b>3</b>	<b>Methods</b>	<b>21</b>
3.1	Optimizing the conventional CNN structure . . . . .	21
3.1.1	Network structure notations . . . . .	22
3.1.2	Used network architectures . . . . .	22
3.1.3	Regularization to mitigate overfitting . . . . .	22
3.2	Implementing Capsule Network . . . . .	25

---

3.3	Faster R-CNN retraining an existing network . . . . .	26
3.4	Explanation of used data sets . . . . .	27
3.4.1	The GTSRB data set for classification . . . . .	28
3.4.2	The GTSDb data set for object detection . . . . .	29
3.5	Software and setup . . . . .	30
<b>4</b>	<b>Results</b>	<b>31</b>
4.1	CNN on par with human accuracy . . . . .	31
4.2	Capsule Network state of the art performance . . . . .	40
4.3	Faster R-CNN shows promising detection but classification proved harder . . . . .	45
<b>5</b>	<b>Discussion</b>	<b>49</b>
5.1	The simple CNN structure is able to find complex features . . . . .	49
5.2	Capsule Network performs well outside of the simple MNIST data set	52
5.3	Capsule Network outperforms CNN at the expense of training time .	53
5.4	Intraclass homogeneity of GTSRB could contribute to the high accuracy	54
5.5	Augmented data lead to great improvement of the Faster R-CNN . .	56
<b>6</b>	<b>Conclusion</b>	<b>59</b>
	<b>Bibliography</b>	<b>61</b>
<b>A</b>	<b>Configuration file for Faster R-CNN</b>	<b>I</b>
<b>B</b>	<b>Additional CNN results</b>	<b>IV</b>



# 1

## Introduction

Recent advances in the field of deep learning have made it possible to solve many previously difficult problems. The most well known examples are perhaps image and speech recognition, where this method has beaten the previous records set by other techniques, however the possible applications are many more. Deep learning has already been successfully applied to such different tasks as predicting the effect of mutations in DNA and analyzing particle accelerator data [1]. With so many applications already demonstrated it is evident that this technique has a huge potential and could be applied to a variety of different problems. Another benefit is that deep learning methods are able to find solutions to problems that are too large and complex for us humans to understand and solve.

One of the strengths of deep learning methods is that they are good at learning from large amounts of data, which has been a challenge for other machine learning models. For deep learning the problem is reversed, a large data set is needed to get a good performance from the network. Larger amounts of data tend to lead to better performance. Jeff Dean, head of the Google Brain team, has said that deep learning methods really starts working well when you have about 100000 examples or more [2]. A serious challenge associated with large data sets is an increasing computational load. Recent improvements to especially graphic processing units (GPUs) and special processing units defined specifically for machine learning, such as Google's TPUs, have contributed to great breakthroughs in this area.

When Geoffrey Hinton et al. published a paper [3] in 2006 that showed how to train a deep neural network with state-of-the-art precision on classifying handwritten digits, the field started to develop rapidly. Today focus has turned to more complex problems, for example developing autonomous vehicles. One of the challenges this industry faces is to reliably interpret information from its surroundings, interpreting traffic signs is one such fundamental task. Deep learning has since 2010 continuously set pioneering records both on the ability to detect [4] and classify [5] traffic signs. Because of their proven strength in the field, this work will investigate and compare several of these methods. Firstly convolutional neural networks that are the current state of the art technique for this problem will be discussed, before moving on to analyzing a new model called Capsule Network that works in a fundamentally different way.

### 1.1 Aim

The overall aim is to investigate different deep learning models and their applications. The focus is on implementing neural network algorithms to classify road signs from the German Traffic Sign Recognition Benchmark (GTSRB) since sign recognition is an important part for autonomous vehicles. By studying the machine learning field of neural networks we hope to achieve the goal to reach an accuracy close to the current highest results. The best result published on the GTSRB data set is 99.71 % accuracy [6], achieved in 2017. In comparison human performance corresponds to an accuracy of 98.84 % [7]. More results are available at the GTSRB web page [8]. In addition to classification, the report also aims to explore road sign detection on the German Traffic Sign Detection Benchmark (GTSDB).

### 1.2 Constraints

While it is possible to apply deep learning to detect and classify many types of objects such as cars and pedestrians, we have decided to only work with traffic signs. The main reason for this is the availability of a big labelled data set for this task. We also deem detecting and classifying traffic signs as a task that seems interesting and sufficiently challenging. The classification task that we will work on is a single image multi-class classification problem with 43 different classes.

We have decided to only use the GTSRB and GTSDB data sets for validating and testing our models, since they are sufficient for our purposes. Due to this we are limited to detection and classification of signs from color images. It is possible to apply similar techniques for detection and classification of objects from video data but we have excluded this from the task.

We have decided to only use deep learning methods, even if there are many other methods that can be used for solving image classification problems, such as Decision Tree Classifications, Nearest Neighbour and Support Vector Machines [9]. The reason for this decision is that using deep learning is the current state of the art method for image recognition tasks.

### 1.3 Social and ethical aspects

The social and ethical implications of this project is not particularly evident at first sight. Implementing an algorithm using deep learning methods does not directly hurt any person nor does it restrict the integrity of anyone. Therefore the question is rather what the intended use of these methods are, what these implementations



can be expected to achieve and what raw data is used. For instance, using deep learning methods to diagnose disease should be beneficial to society, but if it is done using a persons entire DNA it could be viewed as a breach of privacy. Likewise, implementing an algorithm that can identify a person by their face or voice could be beneficial in some cases, but it could also be used as a tool of surveillance. So in conclusion, as the power of deep learning methods increases so does its potential to do both good and bad. It should be mentioned that the networks and data sets used in this project fall in the category of autonomous vehicles, an area which indeed presents some ethical aspects partly due to the fact that accidents in traffic are bound to happen. The networks controlling the vehicle would then directly or indirectly be responsible for the accident, an issue which must be addressed. However, since the scope of this project does not go to such lengths this is not something which must be considered in this thesis, but merely if this work were to be developed further in the future.

One possible societal problem with traffic sign recognition is the possibility for manipulation and tampering with traffic signs that could lead to networks misclassifying the traffic signs. Eykholt et al. showed in their paper [10] that it is possible to trick a deep neural network that a stop sign was a 45 mph-sign with an 85 % accuracy using just strips of white and black tape. Sitawarin et al. wrote in their paper [11] that it is possible, using different pixel transformations, to create an adversarial sign that in a real world scenario would lead to the network classifying the sign with the adversarial label with an accuracy above 95 %. This shows that it could be possible to manipulate traffic signs on a large scale so that self driving cars miss-classify them and take inappropriate decisions. Therefore it is of high importance to develop more robust and secure systems that are resilient to these kinds of adversarial attacks. It can also be possible to reach this effect by just manipulating a small subset of the pixels in a picture, which gives the effect that the picture for a human still clearly represents the original object but the neural network interprets it as something completely different. One way to counteract this is to include adversarial images in the training of the network so that the network can learn how to detect these special types of manipulated images.

## 1.4 Disposition

The report has been divided into theory, methods, results, discussion and conclusion. The theory contains the background needed for understanding the problem and the implementation. The method explains the specific approach to solving the problem and the findings are presented in the results chapter. The last two chapters contain a discussion of the results in relation to the theory and also the final conclusions.

# 2

## Theory

In this chapter the theory that this project is built on is presented. First with a general introduction to how artificial neural networks (ANNs) work and how regularization methods and preprocessing can be used to improve their performance. Then the specific network structures used, CNN, Capsule networks and R-CNN, are introduced. Finally we will present how to evaluate the results.

### 2.1 Artificial neural networks

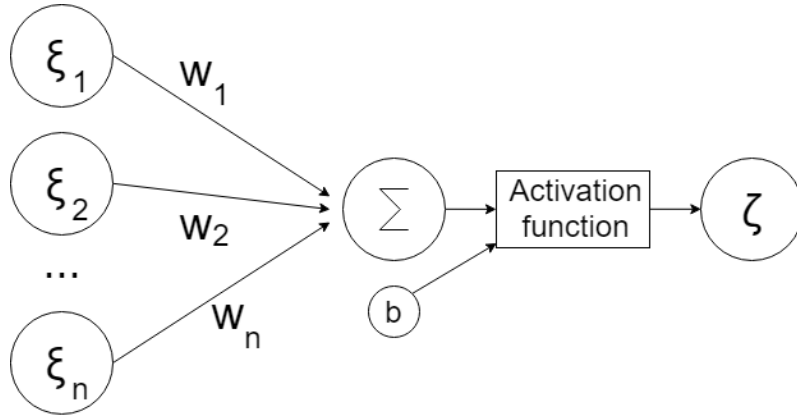
An ANN is a computational model intended to resemble a simplified biological brain [12]. As the name suggests, the model consists of a network of neurons, which can be thought of as nodes. The nodes in the network are connected with weights, through which they communicate. Like the neurons in the brain, artificial neurons can learn information and the learning is related to adjusting the weights in the network [13].

#### 2.1.1 Neurons

Each neuron has one or more inputs and produces after calculations one output. There are several ways to perform these calculations, the most basic being the perceptron [14], see figure 2.1. Let the inputs be represented as  $\xi_i$  for input number  $i$ , which for a perceptron is binary. Each input is then multiplied by a weight  $w_i$  and summed together. After that a bias  $b$  is added. A perceptron delivers a binary output and this is achieved through the introduction of the Heaviside step function (2.1).

$$\zeta = \begin{cases} 0 & \text{if } \sum_i \xi_i w_i + b \leq 0 \\ 1 & \text{if } \sum_i \xi_i w_i + b > 0, \end{cases} \quad (2.1)$$

where  $\zeta$  is the output of the neuron. The perceptron uses a step function to create a binary output but another useful function that Nielsen mentions is the sigmoid



**Figure 2.1:** A schematic drawing of the perceptron.  $\xi_i$  and  $w_i, i = 1, 2, \dots, n$  are the input values and their corresponding weights,  $b$  is the bias and  $\zeta$  is the output.

function

$$\zeta = \sigma(z) = \frac{1}{1 + e^{-z}}, \quad (2.2)$$

where  $z = \sum_i \xi_i w_i + b$ , from (2.1). This is a continuous function delivering an output between 0 and 1. Both the sigmoid function and the Heaviside step function are in this context called activation functions and introduce a non-linearity to the model [15]. Another activation function that is commonly used is the rectified linear unit, ReLU [16]

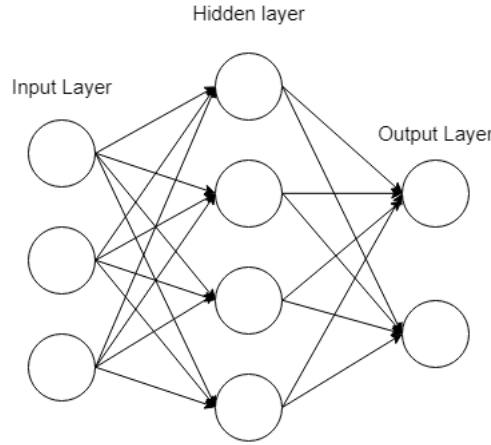
$$g(\xi) = \max(0, \xi), \quad (2.3)$$

where  $g(\xi)$  is the neuron's output, and  $\xi$  is the input to the neuron from the previous layer. When combining multiple neurons, the linear combination of non-linear modules can be thought of as a single multidimensional function. The structure of the network plays a large role in defining the function.

## 2.1.2 Network structure

The neurons of a network can be combined in different ways. A common structure of a neural network is called feedforward network [1]. It consists of multiple layers of neurons, see figure 2.2. The input to the network is processed by the neurons in the first layer, which then feed their calculated results forward to the next layer, and so on. The first and last layers are called the input and output layers. Any layer in between is called a hidden layer. By adding more hidden layers the network can more efficiently solve more complex problems.

Defining a relevant network structure is however not the only problem that must be solved. The weights and biases, or simply parameters, must be assigned in a manner so that the network is able to solve a specific problem. For simple problems in few dimensions, this can be done by hand. However, when dealing with more complex problems another approach of finding the correct parameters is needed. One way is to use supervised learning to let the model adjust its weights .



**Figure 2.2:** Basic feed forward artificial neural network, with an input layer, a hidden layer and an output layer. The information travels from left to right.

### 2.1.3 Supervised learning

Supervised learning is a data driven learning process. The purpose of the learning process is that the model itself should alter its internal parameters, weights and biases, to learn the task at hand [1]. The learning is done by training the model on a data set. In other words, the model is shown data samples together with their desired output. The model is continuously changing its parameters in order learn how to predict the correct output associated with each data sample. In order for the network to evaluate its performance it is necessary to define a measurement of how well the network is performing. This measurement is called the cost function or objective function and is described by Nielsen in his deep learning book as the distance or error between the desired output and the actual output of the network [14].

#### 2.1.4 Cost function

Multiple choices of cost functions are available, one of which is the quadratic cost function [14], also known as the mean squared error. The mean squared error is defined in equation (2.4).

$$C = \frac{1}{2n} \sum_{\xi} \|\zeta - d(\xi)\|^2, \quad (2.4)$$

where  $n$  is the total number of training data,  $\zeta$  is the output vector for input  $\xi$  and  $d(\xi)$  is the desired output [14, 17]. As will be discussed in paragraph 2.1.5, the gradient of the cost functions plays an important role when updating the weights and biases and that updates will be proportional to the gradient. A problem is that when using the quadratic cost function, equation (2.4), and the sigmoid activation function, equation (2.2), the partial derivatives will be proportional to  $\sigma'(z)$  where

$z = \sum_i \xi_i w_i + b$ . This is an issue since  $\sigma'(z)$  tends to zero quite rapidly when  $|z|$  gets large, which will slow down the learning process [14].

The problem with the quadratic cost function can be combated with another choice of cost function. One common choice is the cross-entropy cost function [14], which is defined in equation (2.5).

$$C = -\frac{1}{n} \sum_{\xi} [d(\xi) \ln \zeta + (1 - d(\xi)) \ln (1 - \zeta)]. \quad (2.5)$$

The benefit of this cost function is that the partial derivatives will not be proportional to the derivative of the activation function but rather to  $\sigma(z) - d(\xi)$ , the error in the output [14].

### 2.1.5 Optimizers

Once the cost has been defined the goal of the learning process is to minimize the error for the training data. The most common way of optimizing the cost function is by using gradient descent algorithms. These methods are implemented by step wise updates of the weight and biases. It starts at some initial point along the cost function and then moves in a direction determined with the use of the gradient. Equation 2.6 shows one way of updating the parameters

$$\theta' = \theta - \eta \nabla C, \quad (2.6)$$

where  $\theta'$  is a vector of the updated weights and biases,  $\theta$  is the previous vector of weights and biases,  $\nabla C$  is the gradient of the cost function with respect to weights and biases and  $\eta$  is a positive parameter called learning rate [14]. These steps are updated until some convergence criterion is met. For example until the parameter updates are small. The gradient is either calculated for a subset of data points, this is called stochastic gradient descent (SGD) or for the whole data set at once, called batch gradient descent [18]. The choice of batch size is an important parameter in deep learning optimization. The stochastic gradient descent has the advantage of being easy to implement and it is fast for problems with a large training set. However, it also requires a lot of manual tuning and there is often a problem of finding the right training rate or convergence criteria [19].

The batch gradient have the reverse properties. On the downside it is slower and require more memory and on the upside it is more stable to compute and it is easier to find a good convergence criterion. One significant limitation with batch gradient descent is that the model can get stuck in local minima [20]. The reason is that the method has guaranteed convergence, which means that an update never leads to the cost function increasing.

There are a lot of variations that can be used to improve the gradient descent algorithms. The alterations are used to either improve speed or performance. Examples

of improvements is momentum. This accelerates the algorithm by adding a part of the previous parameter update to the next. It might however decrease performance, so to counteract this one can use something called Nesterov accelerated momentum where the gradient is calculated based on previous parameter values. Another common improvement is to change the learning rate in between the parameter updates. This is done in optimizers such as AdaGrad, AdaDelta, RMSProp and Adam [20].

### 2.1.6 Backpropagation

In order to train a neural network using gradient descent it is necessary to compute the gradient of the cost function,  $\nabla C$ . There are multiple ways to do this, however the computation must be sufficiently fast in order to be useful. A fast algorithm that compute the gradient is called backpropagation. In modern applications of neural networks backpropagation is the dominating method to compute the gradient. The backpropagation algorithm is in essence just an application of the chain rule for derivatives. At the core of the algorithm is the insight that the gradient of each layer can be expressed in terms of the gradient of the next layer [17]. This means that if we are able to determine the gradient in the last layer we can recursively propagate back to compute the gradient of the entire network.

First we start by defining the input to a specific neuron,  $j$ , as

$$x_j = \sum_i y_i w_{ji}, \quad (2.7)$$

where  $y_i$  are the output of the neurons connected to  $j$  and  $w_{ji}$  are the weights associated to these connections. Biases can also be introduced to this input in the form of an extra connection which always have the output 1 [17]. However, in the rest of this section biases will be omitted for the sake of simplicity.

Further, we define the output of neuron  $j$  as

$$y_j = \frac{1}{1 + e^{-x_j}}, \quad (2.8)$$

known as the sigmoid function. It is possible to choose another function in equation (2.8), any function with bounded derivative will suffice [17]. The sigmoid function will remain the definition of the output in this section, but the steps that follow are easy to generalize to other choices.

With these definitions in place we are ready to proceed to calculate the partial derivatives of the cost function, as defined in equation (2.4). The first step is to calculate the partial derivative with respect to each neuron in the output layer, which yields

$$\frac{\partial C}{\partial y_j} = y_j - d_j, \quad (2.9)$$

where  $j$  is a index over all neurons in the output layer and  $d$  is the desired output. Applying the chain rule the partial derivative of the cost function with respect to the input can be expressed as

$$\frac{\partial C}{\partial x_j} = \frac{\partial C}{\partial y_j} \cdot \frac{\partial y_j}{\partial x_j}. \quad (2.10)$$

Using equation (2.8) to compute  $\partial y_j / \partial x_j$  and inserting the result in equation 2.10 the following expression is reached

$$\frac{\partial C}{\partial x_j} = \frac{\partial C}{\partial y_j} \cdot y_j(1 - y_j). \quad (2.11)$$

Equation 2.11 describes how the cost function is affected by changes in the input  $x_j$ . However, the input is a function of the outputs in the previous layer and the weights associated to these connections. Consequently is possible to calculate how the cost function changes depending on these weights and outputs. Using the chain rule and definition in equation (2.7) the following holds.

$$\frac{\partial C}{\partial w_{ji}} = \frac{\partial C}{\partial x_j} \cdot \frac{\partial x_j}{\partial w_{ji}} = \frac{\partial C}{\partial x_j} \cdot y_j \quad (2.12)$$

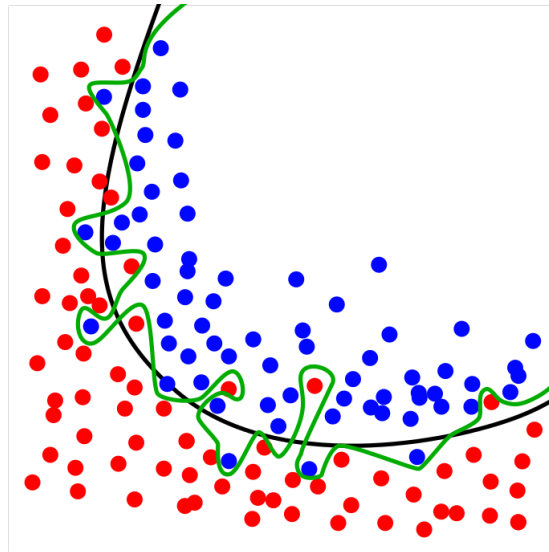
and

$$\frac{\partial C}{\partial y_i} = \frac{\partial C}{\partial x_j} \cdot \frac{\partial x_j}{\partial y_i} = \frac{\partial C}{\partial x_j} \cdot w_{ji}. \quad (2.13)$$

It is clear from the calculations how to compute the partial derivative,  $\partial C / \partial y$ , for each neuron in the second last layer given  $\partial C / \partial y$  in the output layer. By repeating this process for each preceding layer until reaching the input layer and applying equation (2.12) for each weight in the network, an expression is obtained for the gradient of the cost function,  $\nabla C$ .

## 2.2 Regularization

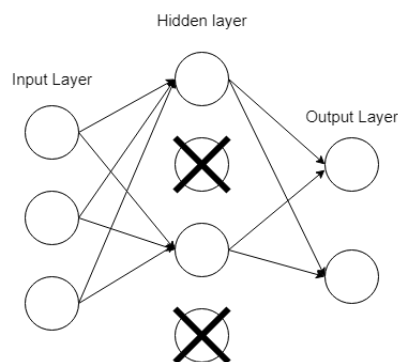
One of the main challenges in machine learning is overfitting, which is when a model adapts too much to the training set and loses its ability to generalize to data [21]. The model learns to find patterns in the noise that can only be found in the training data and therefore learn things that do not help it generalize to new data [22]. An example of overfitting can be seen in figure 2.3. There are three main ways to reduce overfitting, which are using more data, reducing the noise in the training data and lastly, constraining the model so that it becomes simpler. Constraining the model in this way is called regularization [21]. Two regularization methods that are often used in deep learning are dropout and batch normalization [23].



**Figure 2.3:** The figure shows two models that have been trained on the data points. The green line represents an overfitted model and the black line represents a regularized model. While the green line best follows the training data, it is likely that the black line will better generalize to new unseen data. From [24]. CC BY-SA 4.0

### 2.2.1 Dropout

To implement dropout means to randomly "kill" some neurons, i.e. temporarily set the output value to 0, during training. The killed neuron will not be able to pass information, visualized in figure 2.4. This has the effect that the network cannot depend too heavily on individual neurons since they sometimes do not work properly. This has been proven to increase networks ability to generalize and thereby decreasing overfitting [25].



**Figure 2.4:** Visualization of how dropout works. Two neurons in the hidden layer are dead, which means they are set to zero. Thus no information will be able to pass through these neurons and the network has to use other paths.



### 2.2.2 Batch normalization

A problem with image recognition is the fact that the picture may vary in intensity, which would be noticeable all through the network. The phenomenon when the distribution of input changes but the distribution of the output remains the same is called covariate shift [26]. A way to handle these different input distributions is using Batch Normalization [27]. Since each mini-batch is different, so are their distributions. Batch Normalization takes an input, usually a defined batch size, calculates the variance and the mean and normalizes the data such that the mean is 0 and the variance is 1. Batch Normalization can be used between different layers inside of the network.

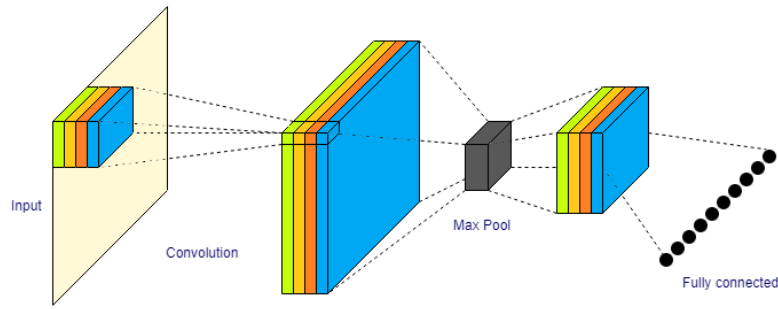
## 2.3 Image preprocessing and data augmentation

In order to improve a data set for machine learning purposes, data augmentation can be used [28]. Data augmentation is based on increasing the size of a data set by generating new examples from existing ones. The new examples can be generated by adding different transformations, for example rotation, pixel value shifting, mirroring and shading, to the original examples. Data augmentation can counteract overfitting by working as a regularizer.

A preprocessing tool is cropping, which is removing the background of the image by only retaining the object of interest itself. This reduces the amount of noise in the image, which can make it easier to classify. A combination of cropping and resizing makes the data set consist of equally sized images, which only portraits the relevant object.

## 2.4 Convolutional Neural Networks

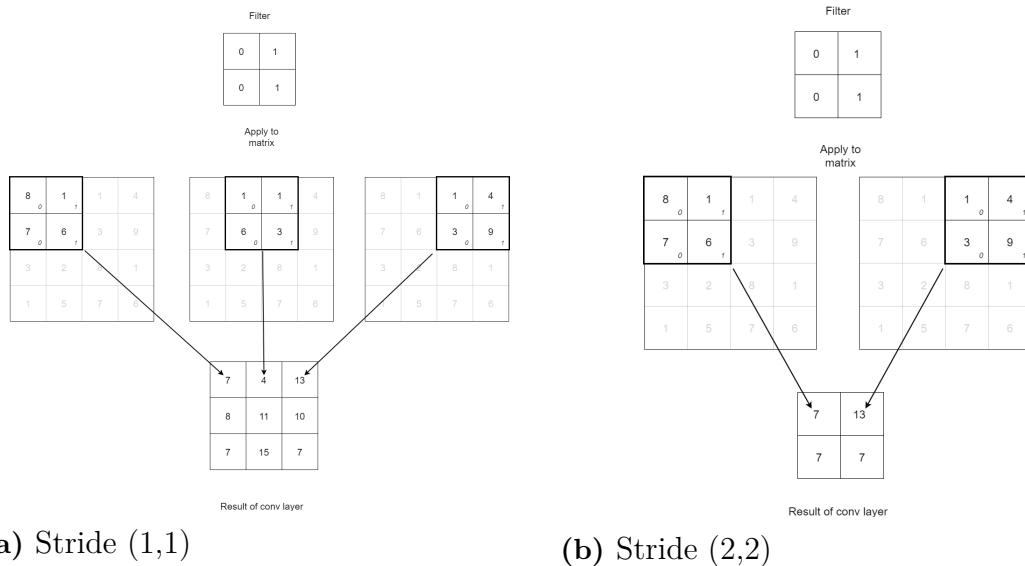
A neural network architecture that has proven very effective at classifying images is the convolutional neural network [29], and example is shown in figure 2.5. One motivation for using CNN is that they are invariant to scaling and translation of features, to some extent. It is achieved by their ability to use the knowledge the network has learned at all locations in the image. Another reason is that they are able to combine basic features such as oriented edges and corners into higher level features.



**Figure 2.5:** A visualization of the structure of a CNN with a conv layer containing 4 filters, followed by a max pooling layer and a fully connected output layer with 10 neurons.

### 2.4.1 Convolutional layers

The characteristic building block is the convolutional layer [30]. The convolutional layer, or conv layer, consists of a number of small filters. A filter is a rectangular block of weights. Each filter has its own set of weights and bias. The calculations are performed by letting the filter slide over the input and for each position calculate an output value, see figure 2.6a. The output is produced as described in section 2.1.1 by summing the element-wise multiplications and possibly adding a bias. An activation function is then applied where the ReLU function, see equation 2.3, is a common choice [16]. A reason is that the ReLU activation function provides faster learning and is cheaper to compute than e.g. the sigmoid function [31].



**Figure 2.6:** Depictions of the application of a (2,2) filter to an input, which represents the operations of a convolutional layer. The activation map is composed of the output values, which are calculated for each filter position by performing element wise multiplications and summing the results. Figure 2.6a and figure 2.6b corresponds to stride (1,1) and (2,2).

The output for each filter will be a two-dimensional activation map. Thus the total output is three-dimensional if there is more than one filter. The size of the output can vary depending on how many steps, or strides, the filter is moved after each calculation. In figure 2.6a a (1,1)-stride is used, which means that filter is shifted one step at a time. A (2,2)-stride would result in the filter moving two steps instead, which can be seen in figure 2.6b.

After successful supervised training the filters will start to react to certain features [30]. For example one filter might create large outputs when covering straight edges in an image, while another might react to the color blue.

### 2.4.2 Max pooling

Another building block that is used in convolutional layers is the max pooling layer [30]. It works similarly to the conv layer in the sense that it consists of a filter. However this filter does not perform any matrix calculations. It simply outputs the largest value of the input that the filter covers, as can be seen in figure 2.7. The filter is then applied to the rest of the input step by step. This technique makes the convolutional network less sensitive to where in the image a feature is presented [32]. At the same time a lot of information is lost.

$$\begin{bmatrix} 2 & 4 & 1 & 4 \\ 1 & 5 & 0 & 1 \\ 8 & 3 & 5 & 4 \\ 1 & 0 & 1 & 2 \end{bmatrix} \rightarrow \max \left( \begin{bmatrix} 2 & 4 \\ 1 & 5 \end{bmatrix} \right) = 5 \rightarrow \begin{bmatrix} 5 & 4 \\ 8 & 5 \end{bmatrix}$$

**Figure 2.7:** Max pooling done with a (2,2) filter and a (2,2) stride. The max pooling will take the highest value in each stride and create a new down sampled matrix.

### 2.4.3 Fully connected layers

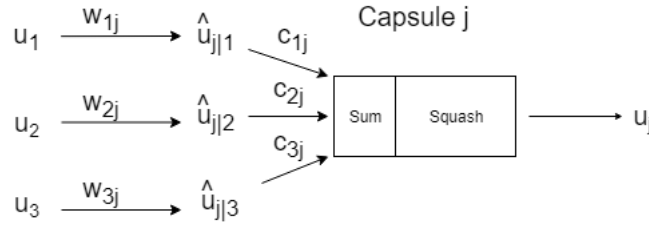
Convolutional networks usually start with conv layers and max pooling layers followed by fully connected layers [30], which have the structure of an ordinary feed forward network. The last layer, or output layer, of a network with the task to classify an image will have as many output neurons as there are possible classes. The activation function for the last layer is the softmax function [33],

$$g(\mathbf{x})_j = \frac{e^{x_j}}{\sum_{k=1}^N e^{x_k}}, \quad j = 1, \dots, N \quad (2.14)$$

where  $g(\mathbf{x})_j$  is the output of the  $j$ :th neuron, and  $x_j$  is the input to the  $j$ :th neuron. The softmax function produces a vector with values between 0 and 1 for each class with the total sum of 1. This value can be interpreted as how confident the network is that the input belongs to each class [34].

### 2.4.4 Capsule Network

Capsule Networks are a new network architecture that was first introduced in October 2017 by Sabour et al. [35]. The theory for the networks was developed further in a paper published by the same authors in the beginning of 2018 [36]. One of the main difference between Capsule networks and earlier network architectures is that they contain a new building block called capsules [37]. A simple way to describe capsules are to say that they are a group of neurons. Capsules add another dimension of structure in the network. Neurons build capsules, capsules build layers and layers build networks. As mentioned in section 2.1.1, a neuron takes multiple scalars as input and outputs one scalar. This is different in the capsules since they handle vectors instead of scalars. Similarly to the neuron, the capsule can for example represent an entity in an image. As the capsule has vectors, the dimensions of the vectors can encode poses of the entity [35]. The poses can for example be position, contrast or deformation. One common way of visualizing the advantages of a capsule network is by looking at an image of a face with the eyes placed on the chin and the mouth placed on the forehead. A basic CNN has no built in structure that cares about the relation between the eyes, mouth and face. A capsule network handles these relations. Consider the case when a specific capsule in a layer represents the nose entity and a capsule in the next layer represents the face entity. The face capsule then predicts where the face should be in relation to the nose. If that prediction agrees with the actual position of the face, and all other face entities also agree, then the capsule activates. Capsule networks also handle different viewpoints this way and that is one of the reasons that capsule networks generalizes better than CNN [35].



**Figure 2.8:** Calculations in a capsule. The input vector  $\mathbf{u}$  is multiplied with a transformation matrix  $\mathbf{W}_{ij}$  thus producing a prediction vector  $\hat{\mathbf{u}}_{j|i}$ . Then the prediction vector is multiplied with the scalar coupling coefficient  $c_{ij}$ . The vectors are then summed and the squash function is applied to produce an output with length between 0 and 1.

The mathematical implementation of the capsules is shown in figure 2.8. The total input  $\mathbf{s}_j$  is calculated as a weighted sum of the prediction vectors  $\hat{\mathbf{u}}_{j|i}$  from the previous layers. The prediction vectors are calculated by multiplying the output vector  $\mathbf{u}_i$  from the previous layer with a weight matrix  $\mathbf{W}_{ij}$ ,

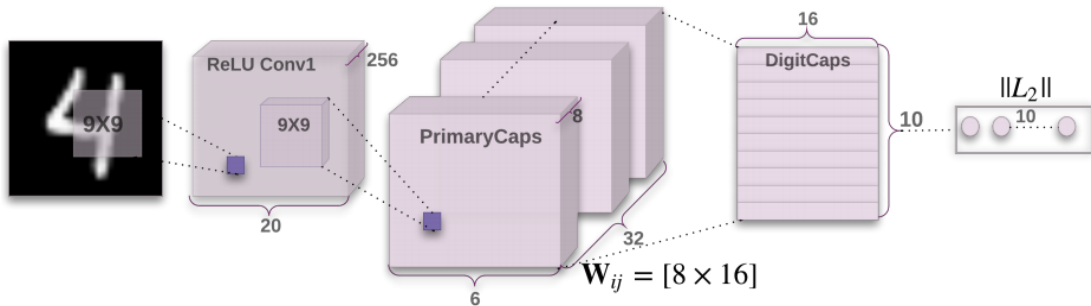
$$\mathbf{s}_j = \sum_i c_{ij} \hat{\mathbf{u}}_{j|i}, \quad \hat{\mathbf{u}}_{j|i} = \mathbf{W}_{ij} \mathbf{u}_i. \quad (2.15)$$

where  $c_{ij}$  are coupling coefficients that are calculated by something called dynamic routing, which will be explained later. The weight matrices  $\mathbf{W}_{ij}$  are learned by performing back-propagation. A non-linear activation function is applied to the total input to the capsule to make sure that the output vector from the capsule has a length that can be interpreted as a probability. The function that is used is a squashing function that ensures that short vectors get their length transformed to be close to zero while long vectors get their size transformed to be slightly under 1, as follows

$$\mathbf{v}_j = \frac{\|\mathbf{s}_j\|^2}{1 + \|\mathbf{s}_j\|^2} \frac{\mathbf{s}_j}{\|\mathbf{s}_j\|}, \quad (2.16)$$

where  $\mathbf{v}_j$  is a vector output from capsule  $j$  and  $\mathbf{s}$  as described in equation (2.15).

In conventional CNNs the pooling layers are used to ensure invariance in the network, i.e. a small change in input does not change the output [37]. The capsule network does not implement pooling. This is due to the fact that the pooling loses valuable information about the pose of the entities. To solve the problem of invariance, dynamic routing is introduced. Dynamic routing is an algorithm resulting in the coupling coefficients  $c_{ij}$  seen in figure 2.8. Max pooling in CNN essentially means that the neurons in a deeper layer look at a group of neurons in the previous layer and take into account the ones that produce the highest output and disregarding all other. In comparison, the dynamic routing sends the information from a lower layer capsule to the capsule in the next layer that can account for that information. To give an intuition behind this the following example can be examined. A circle shape can be sent to a capsule for a bicycle but it is not going to be sent to a boat capsule. This is done by the coupling coefficients that are trained by the dynamic routing algorithm. In short, the algorithm looks at what lower level capsules that agree tightly, i.e. has the same direction of the output vector, and assigns large values of the coupling coefficients to those input vectors that have the same direction [35]. The capsule network shown in figure 2.9 consists of one convolutional layer and two



**Figure 2.9:** Example of a Capsule Network structure used to classify digits. It consists of one initial conv layer, two capsule layers and a decoder network at the end. From [35]. Reproduced with permission.

capsule layers, PrimaryCaps and DigitCaps, and is designed for classification. The convolutional layer extracts features which the first capsule layer then combine to be able to detect more complicated structures. The dynamic routing algorithm is run between the two capsule layers. It is the length of the output vectors from the final

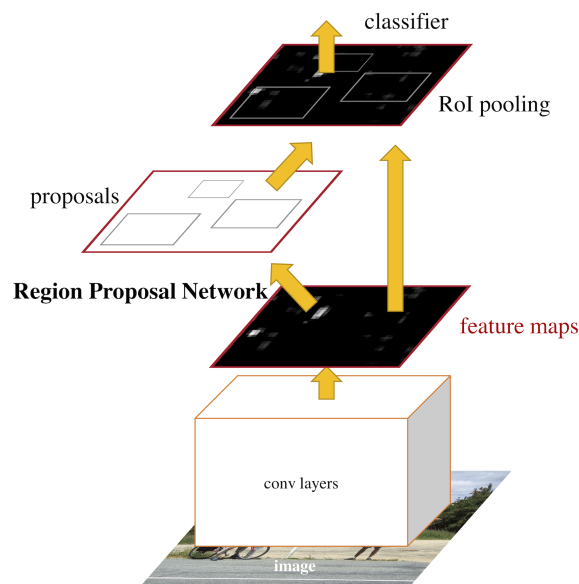
capsule layer that describe the probability that the input belongs to the different classes and the direction of the vector describes the different poses of the image.

One regularization technique usually used in Capsule Networks is to use the activity vector from the correct digit capsule to reconstruct the input image [35]. This is done by using a decoder that consists of fully connected layers that try to reproduce the image from the activity vector. This reconstructed image is then used to calculate a reconstruction loss, this loss is scaled down to ensure that it does not dominate the classification loss while training. The reason for introducing this reconstruction loss is to encourage the capsules in the last layer to encode the information needed to recreate the digit.

One restriction with Capsule networks is that they take longer time to train than CNN and other more conventional network structures, but this can in some cases be outweighed by the benefit they give by being able to produce better results with less training data [35].

## 2.5 Networks for object detection

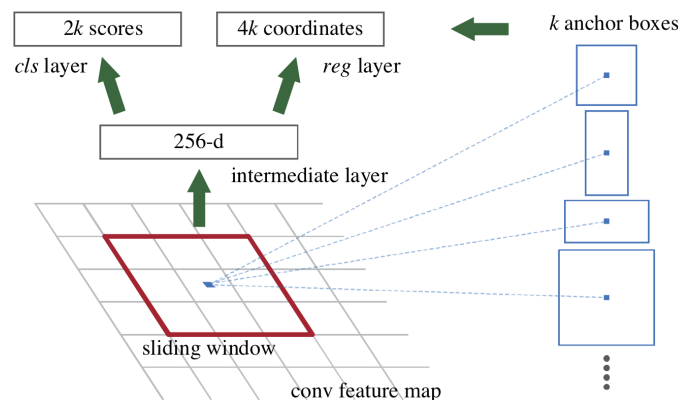
For tasks that only involves classification using networks, such as CNNs and Capsule Networks, works well. However, for tasks that involve detection, that is predicting the location of an object, as well as classifications there is a need for other designs. This section introduces new types of networks that meet these requirements.



**Figure 2.10:** Illustration over the structure of Faster R-CNN showing how the classification and region proposal networks shares the same feature map. From [38]. ©2017, IEEE. Reproduced with permission.

The most straight forward way to tackle the object detection problem would be to take advantage of the CNN and feed that CNN with a bunch of random regions from the original image. The proposed regions could either be hard coded or chosen by some object proposal algorithm, for instance selective search [39]. This is the foundation behind the method called Regions with CNN features (R-CNN). In addition to a CNN the R-CNN uses a couple of non-ANN techniques for classification and tightens the bounding box, namely a support vector machine (SVM) and linear regression [39]. R-CNN fulfills all requirements mentioned above, the only problem is that this model is rather slow.

To increase the speed a modified version called Fast R-CNN was introduced in a paper from 2015 [40]. As described by this paper a Fast R-CNN mainly differs from a regular R-CNN in two ways. First of all, instead of passing each proposed region separately through the CNN the image is just passed once as a whole and then all region proposals shares the same feature map. For each proposed region the features are then extracted by selecting the corresponding region from the shared feature map. The SVM is also replaced with a softmax layer after the CNN and the linear regressor is replaced with an ANN regressor layer. This means that all outputs are computed by a single network, instead of three different parts as with the R-CNN [40].



**Figure 2.11:** Figure illustrating how sliding windows and anchors are used in the region proposal network to generate proposed regions of interest. The *cls* layer scores each proposal and the *reg* layer regresses the bounding box. From [38]. ©2017, IEEE. Reproduced with permission.

Another improvement to the R-CNN-based networks is called Faster R-CNN, which removes the selective search used by the earlier version [38]. By using the features already calculated by the CNN to make region proposals, instead of a separate method looking at the original picture, speed was increased further. This means that Faster R-CNN uses the same feature map for both region proposals and classifications. The general structure is illustrated in figure 2.10. The region proposals are generated by what is called a Region proposal network (RPN) that both generates suggested bounding boxes and scores how likely each box is to contain an object. The RPN is a fully convolutional network that slides a window over the feature map and for each window position the network gives  $k$  possible bounding boxes. These  $k$  proposals

for each position are predefined and called anchors. Each anchor is centered in the middle of the sliding window and is defined by an aspect ratio and a scale. This means that each of the  $k$  anchors is a box with a specific aspect ratio and size, selected to match what shapes the object of interest are expected to have. The region proposal network is illustrated in figure 2.11. For each proposed box from the entire feature map there is an associated score, indicating how likely the network thinks it is that this box will contain an object of interest. These suggested regions are then passed to a Fast R-CNN that will give the final output, bounding boxes, classes and confidence scores.

## 2.6 Evaluating the results

There are several ways to evaluate the results from a neural network. One way is to define a performance metric which assigns a number to the performance of the network. Several common performance metrics will be introduced as well as two ways of visually evaluating the performance of a network.

### 2.6.1 Precision, Recall and $F_1$ -score

For data sets with class imbalance, large variations of the number of examples for each class, it can be problematic to use accuracy as the only performance metric [41]. The reason for this is that it tends to undervalue how well classifiers are doing on smaller classes. The reason for this is that it is calculated by dividing the total number of correct classifications with the total number of classifications, and therefore doesn't take into consideration that the size of the classes can differ. To counteract this the two performance metrics precision and recall can be used to calculate how well the model performs on each class. Precision is defined as the percentage of predictions that are correct and recall is defined as the percentage of positive instances that are correctly classified. Increasing one of these measures leads to the other one being reduced, so a suitable balance between these two measures has to be found. A convenient way to do this is to combine them into a single metric called the  $F_1$ -score. The  $F_1$ -score is defined as the harmonic mean of precision and recall, using the harmonic mean ensures that a large  $F_1$ -score is only achieved when both recall and precision are high. It is defined as

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$



### 2.6.2 Mean average precision

When working on a task that involves both segmentation as well as classification it is not as clear how the performance of the network should be measured. For instance, it is not self evident how a percentage of correct classifications should be defined. One measurement often used for classification and segmentation networks is called mean average precision (mAP). The exact definition can vary slightly, however in this report the definition from Pascal VOC is used and it is described below [42].

Precision and recall are two measurements used to compute mAP and the definition is presented in section 2.6.1 with the addition that in a classification and detection task it is not enough that the predicted class is correct. The overlap between the predicted and actual bounding boxes must also be sufficiently large. The actual bounding box of an object within an image is usually called the ground truth bounding box. Intersection over union (IoU) is the measurement used in the Pascal VOC definition and it must be larger than 0.5 for a prediction to be considered true. The mathematical definition of IoU is as follows

$$\text{IoU} = \frac{\text{area}(B_p \cap B_{gt})}{\text{area}(B_p \cup B_{gt})}, \quad (2.17)$$

where  $B_p$  is the predicted bounding box and  $B_{gt}$  is the ground truth box.

The average precision (AP) is then defined as the mean precision at eleven different recall levels.

$$AP = \frac{1}{11} \sum_{r \in \{0.0, 0.1, \dots, 1.0\}} p_{\text{interpol}}(r), \quad (2.18)$$

where  $p_{\text{interpol}}$  is the interpolated precision of a given recall level, that is if there is a higher recall level with better precision that value is used instead. The definition of this is

$$p_{\text{interpol}}(r) = \max p(\tilde{r}), \quad \forall \tilde{r} > r \quad (2.19)$$

where  $p(\tilde{r})$  is the precision corresponding to recall  $\tilde{r}$ .

The average precision is computed for each class in the test set and our desired measurement, mAP, is simply the mean of the average precision for each class.

### 2.6.3 Confusion matrix

In a machine learning problem with many classes to classify it can be a good idea to see if a class is recurrently mistaken for another. This is visualized by the confusion matrix [41]. The confusion matrix is a  $n$  by  $n$  matrix, where  $n$  is the number of classes. It shows for each column which output is correct and for each row the output of the network. In a perfectly trained network, one would expect every element of the confusion matrix to be zero except the non-zero diagonal elements. This would mean that all data had been correctly classified [43].

### 2.6.4 Class clustering

There are numerous problems with evaluating and visualising a neural network. A neural network normally has a lot of parameters. To visualize how a network classifies is thus a multidimensional problem. For example: To be able to visualize the content of a 1000 dimensional vector, one needs to de-dimensionalize the 1000-element vector into a 2- or 3-D vector. There are several methods for doing this e.g. principal component analysis [44] (PCA) and t-distributed stochastic neighbor embedding (t-sne), the latter requires that PCA has been used first to reduce the number of dimensions partially to be able to converge [45]. Using a dimensionality reduction technique allows the user to cluster an intangible amount of information into something interpretable.

It is also possible to gain some intuition on the information in a high dimensional room by using a K-Nearest Neighbor classifier (K being an arbitrary positive number). This technique calculates the norm of the difference between a given input vector and every other vector in the data set. The K vectors corresponding to the lowest norm - the so called nearest neighbors - are then used for predicting the class of the input vector. The predicted class is calculated as the mode of the classes of the nearest neighbors [15].

# 3

## Methods

In this chapter our approach to solving the problems is presented starting with a description of how our networks were designed. The software and computational resources used are later described. While trying to obtain good results for the GT-SRB data two different network structures, a regular CNN and a Capsule Network were implemented. Each network had its own preprocessing and data augmentation developed. Furthermore this chapter also describes the implementation of a Faster R-CNN to solve the detection problem.

### 3.1 Optimizing the conventional CNN structure

In order to find a network structure well suited for the GTSRB data set we decided to start with a simple model architecture, which would be improved iteratively part by part. The CNN MNIST Classifier<sup>1</sup> is a network that has been proven to work on the MNIST data set, a set of grey scaled handwritten digits [46]. With regularization the CNN MNIST Classifier has achieved over 97% accuracy on the test data set from MNIST. The architecture of the network is based on the following pattern:  $C_5(32) - P_2^2 - C_5(64) - P_2^2 - D(1024) - D(10)$ , the notation is described in section 3.1.1. The network was trained on the GTSRB data set in order to confirm that it is able to learn how to classify traffic signs. The result was a 91.6% accuracy on the test set after 40 epochs training and with a batch size of 200, which lead us to believe that a similar structure could perform well after tuning. When training a CNN, 20% of the training set was randomly selected to be used as validation set.

---

<sup>1</sup>TensorFlow Tutorial describing the network: <https://www.tensorflow.org/tutorials/layers> (available May 14 2018)

### 3.1.1 Network structure notations

To be able to write the structure of the networks in a compact manner, we have created a notation that describes each layer.

$$C_{\text{filter size}}^{\text{number of strides}} (\text{number of filters/neurons})$$

The letter indicates what kind of layers it is: C-convolutional, P-Pooling, D-dense (fully-connected). The superscript is omitted if number of strides is equal to one. The filter size is always quadratic with same depth as the depth of the input to the layer. The number of filters/neurons is the number of filters used for a conv layer and the number of neurons used for a dense layer.

A simple example can be visualized in figure 2.5. A structure of a CNN can be denoted  $C_5(4) - P_2^2 - D(10)$ . It is a network with a convolutional layer as the first layer with a (5,5) filter, with stride (1,1) and 4 different filters. Followed up by a pooling layer with a stride (2,2) and (2,2) filter. The last layer is a dense layer (fully-connected) with 10 neurons.

### 3.1.2 Used network architectures

Eight different networks similar to the CNN MNIST Classifier were created (see Table 3.1). Due to limited computing resources deeper neural networks with even more convolutional layers could not be trained efficiently. Network structures with more than four conv layers were consequently refrained from. Since some of these networks still are larger than the CNN MNIST classifier they are more computationally expensive. In order to avoid memory issues smaller batch sizes were used during training. Every network was trained three times with different batch sizes; 5, 10 and 20. The reason for this was to receive more information of the networks performed. The chosen optimizer was SGD with a learn rate of 0.01 and the loss function used was categorical cross-entropy. Each training was interrupted when the validation loss stopped decreasing.

Network 4 in table 3.1 performed the best out of the eight different networks and was selected for further improvement.

### 3.1.3 Regularization to mitigate overfitting

Our intention was to increase the ability of the network to learn how to generalize, i.e. learn the features of the different classes rather than features belonging to the training set. For that reason batch normalization and dropout were implemented in network 4. Each regularization method was allowed to be used on either the conv

**Table 3.1:** Different variations to the architecture of the CNN MNIST Classifier. Specifically there are variations to filter sizes and layer configurations.

Network number	Architecture
1	$C_5(16)-P_2^2-C_5(32)-P_2^2-D(1000)-D(500)-D(43)$
2	$C_5(16)-P_2^2-D(500)-D(43)$
3	$C_5(16)-C_5(32)-P_2^2-C_5(64)-C_5(128)-P_2^2-D(1000)-D(500)-D(43)$
4	<b><math>C_5(16)-C_5(32)-P_2^2-C_3(64)-C_3(128)-P_2^2-D(1000)-D(500)-D(43)</math></b>
5	$C_3(16)-C_3(32)-P_2^2-C_3(64)-C_3(128)-P_2^2-D(1000)-D(500)-D(43)$
6	$C_5(16)-C_5(32)-P_2^2-C_5(64)-C_5(128)-D(1000)-D(500)-D(43)$
7	$C_4(16)-C_4(32)-P_2^2-C_4(64)-C_4(128)-D(1000)-D(500)-D(43)$
8	$C_3(16)-C_3(32)-P_2^2-C_3(64)-C_3(128)-D(1000)-D(500)-D(43)$

layers, the fully connected layers, the whole network or not at all. In order to find the best combinations 16 different networks covering all possible structure variations were defined. Each network was trained repeatedly with different dropout rates (see Table 3.2).

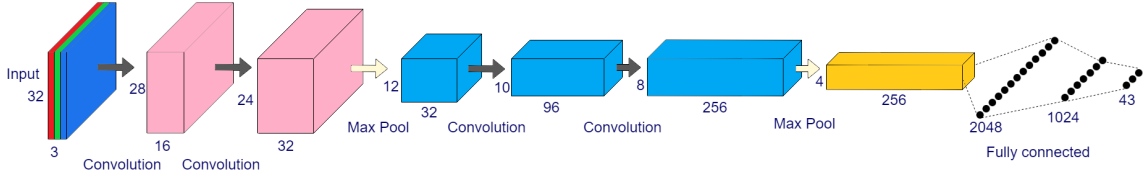
**Table 3.2:** Parameter values for training different iterations of network 4 from Table 3.1.

Batch size	Epochs	Optimizer	Learn rate	Loss function	Dropout rate
10 / 15	30	SGD	0.01	Categorical cross-entropy	0.2 / 0.4 / 0.6

Following the patterns observed more tests were conducted. The two different regularization techniques seemed to have different effects on the performance of the network depending on where in the network they were applied. We chose to restrict how the regularization methods could be applied and focused on the configurations that performed best in the previous test. On that account batch normalization was limited to the conv layers and dropout was only allowed in the fully connected layers. The four network variations that are viable with these constraints were trained with batch size 15 for 10 epochs, first using 50% dropout with 0.01 learn rate and then 40% drop out with 0.02 learn rate. No other change was made to optimizer or loss function. The next step would be to chose one model and train it extensively using a smaller learn rate, in order to let the learning process carefully try to find optimal values of the parameters of the network. We decided to use the model which made use of both batch normalization and dropout. The reasoning behind this was that prolonged training can lead to the model overfitting [47]. Since the application of these methods did not seem to lower the overall performance of the network, other than decreasing the learning speed, there was no disadvantage in using it.

A comprehensive training was then performed on the network with both batch normalization and dropout active. The training time was increased to 100 epochs and the learn rate was lowered to 0.005. The batch size was kept at 15 and the dropout rate was slightly lowered to 37%. The reason for lowering the dropout rate was that a previous network, which was trained with a dropout rate of 40%, did not show

### 3. Methods



**Figure 3.1:** The structure of the final network. Each box represents the dimension of the data at every layer and the arrows refer to the operations done by the layers of the network. Dropout and batch normalization has not been visualized.

signs of overfitting. Therefore we decided to lower the strength of the regularization used.

Results showed that among the missclassified images some classes were over-represented. In order to further increase the performance we started using data augmentation to enlarge the training set and provide the network with more data, especially of the classes with fewer original images. The network was trained using the extended data set, see section 3.4.1. Due to more data per epoch, the training time was lowered to 50 epochs. Remaining parameters were unchanged.

Since the model was training on a larger data set, the risk of it overfitting was reduced [21]. Therefore we tried to increased the size of the network to see if it was able to generalize better with the help of more neurons. Additional filters were added to some conv layers and the number of neurons in the fully connected layers was increased. This network, see figure 3.1, can be expressed as

$$C_5(16) - C_5(32) - P_2^2 - C_3(96) - C_3(256) - P_2^2 - D(2048) - D(1024) - D(43).$$

Batch normalization is applied after each conv layer and dropout is present before every fully connected layer. Information regarding the training of this network can be seen in Table 3.3. Lastly, the model was trained several times in order to obtain multiple results. Consistent results would strengthen the argument that the model performs well on the data.

**Table 3.3:** Parameter values for training different iterations of a  $C_5(16)$ - $C_5(32)$ - $P_2^2$ - $C_3(96)$ - $C_3(256)$ - $P_2^2$ - $D(2048)$ - $D(1024)$ - $D(43)$  network

Batch size	Epochs	Optimizer	Learn rate	Loss function	Dropout rate
15	75	SGD	0.005	Categorical cross-entropy	0.37

## 3.2 Implementing Capsule Network

A Capsule Network implementation in Keras made for MNIST[46] and based on Hinton’s paper [35] was used as a starting point<sup>2</sup>. The only difference is that the size of the input images is changed to fit the GTSRB dataset and that there now is 43 digit capsules to handle the 43 different classes. This meant in the encoder having one initial convolutional layer followed by two capsule layers, as well as a decoder network with three fully connected layers to handle reconstruction regularization. The architecture is displayed in figure 2.9. To be able to systematically validate the results of our network, 20% of the training data was taken to be used as a validation set. The used network needed all input images to be of the same size, so since the images in GTSRB had varying sizes they had to all be re-sized to the same size. By taking into account the distribution of image sizes and computational load it was decided to resize all images to 32x32. There were large contrast differences in the images. To counteract the negative effects this has on the result contrast normalization was used.

To increase the performance of the network different changes were systematically done to the network, changing one part at a time, to ensure that the reason for the increased result could be clearly understood. The training time was between 4-6 hours, mainly since the network had approximately 21 million learnable parameters, which restricted the number of changes that could be tested. It was decided to not perform any initial data augmentation. The main reason was to investigate the property that Capsule Networks are supposed to be able to better generalize from smaller data sets than conventional methods.

The first thing examined was mini batch sizes and number of epochs. By trying different batch sizes it was found that an optimal batch size was around 30, for following training runs a batch size of 30 was used. Running 40 epochs ensured that the network had converged, but it might lead to overfitting so if the validation accuracy was higher for an earlier epoch this result was used instead. In some cases we reduced the number of epochs to reduce training times, by running multiple different settings with lower number of epochs it is still possible to test their relative performance. By then running the best alternative for higher number of epochs it is still possible to get the optimal result.

Initially there was a small difference between the validation accuracy and the training accuracy. The performance on the test set was however significantly lower than on the validation set. A way to reduce the gap between the validation and test accuracy can be to increase the size of the validation set. Since the gap between the validation and test accuracy was the largest we started by trying to increase the size of the validation set by doing 40/60 and 25/75 validation/training splits, but both of these led to lowering the test accuracy.

---

<sup>2</sup>GitHub repository for the implementation. <https://github.com/XifengGuo/CapsNet-Keras> (available May 14 2018)

The optimization algorithm used at the start was Adam. We decided to try different optimization algorithms since this leads to different speeds of convergence and finding minimas with different accuracies. To find the optimal one the optimization algorithm was changed but everything else was kept exactly the same. The different optimization algorithms that were tried were SGD, SGD with momentum, SGD with nesterov momentum, Adam, AdaGrad, AdaDelta, RMS-prop and Nadam. The one that gave the highest test accuracy was RMS-prop.

Our network still overfitted, the best ways to counteract this is to use regularization techniques. Regularization techniques work especially well to reduce the gap between training and validation accuracy. The main techniques that were implemented initially were batch normalization and dropout at different places in the network. We started by introducing a 50% dropout after both fully connected layers but this lead to a small reduction in the test accuracy. Introducing batch normalization after the first convolutional layer, and after two first fully connected layers lead to an improved result.

We then performed improved preprocessing in MATLAB instead of in Python. We still only performed normalization but were able to implement it more effectively. Then it was tested to use ELU [23] and SELU [48] activation functions instead of using the ReLU activation function.

Finally it was decided to perform more data augmentation to try and increase the performance even further. One of the benefits with Capsule Networks is that they are supposed to be able to perform better with lower amounts of data, but the goal is now really pushing the network and wanting to reduce in essence almost all errors. Since the distribution of classes was skewed it was decided to look at the F1-score to see which model performed better. By looking at the F1-score for the different classes it was possible to see which of the classes that were most commonly misclassified. Using data augmentation to bolster the number of instances of these classes was attempted to improve the performance for these classes. The data augmentation we used was by rotating all weak classes as well as horizontal and vertical flipping for classes with fitting symmetries. To improve the results even further the average of the output over different training runs using different data augmentation. This is called ensemble learning [5] and often increases the accuracy of the classifier as long as the predictors are independent, meaning that they make different types of errors.

### 3.3 Faster R-CNN retraining an existing network

While our Capsule Network and conventional CNN are good at classifying specific classes they lack the ability to find an object within an image and mediate the objects position. For this reason a different kind of network, namely the Faster R-CNN, was selected for implementation. Since the Faster R-CNN is a complicated neural network, it was decided to use an API provided by Tensorflow, the Tensorflow



Object Detection API, which provides scripts for training and evaluation as well as predefined and pretrained networks<sup>3</sup>. For the GTSDb task an implementation called "Faster R-CNN inception v2 coco"<sup>4</sup>, which is a Faster R-CNN trained on the COCO data set, was chosen. This means that the network initially was able to detect and give positions and classes for the objects that belong to the classes included in the COCO data set, e.g. cats and humans [49]. For the network to be able to detect and classify the traffic signs, it had to be retrained on the GTSDb data set, this is called transfer learning [50]. The main reason for selecting this particular implementation is that it is the fastest implementation available from TensorFlow's object detection API. Very few alterations were made to the hyperparameters of the network as we expected that our task was similar enough to the COCO task. However, since all bounding boxes in our task are approximately squares, we removed all other aspect ratios when generating anchors. It was also necessary to change the number of classes to match our task, that is the number of neurons in the output layer. The entire configuration file is included in appendix A.

The network was initially trained on the original training set up to 150000 steps. Because of the relative small size of the original training set it was decided not to split the training set into training and validation sets. Therefore evaluation was only performed on the test set.

After 150000 steps we could draw two conclusions, the mAP appeared to be saturating and we could determine from the evaluation that the detection of signs worked very well but the classification often was incorrect. We therefore decided to take two actions to improve the performance. The training set was extended using the method described in section 3.4.2 and in order to preserve what the network had learned so far, the training was continued with the extended set on the network that was saved after 150000 steps. The network was trained on the extended set for an additional 100000 steps. As another effort to improve the performance a variant that combined Faster R-CNN with the best performing CNN network trained on the GTSRB set was created. This variant used the Faster R-CNN to generate bounding boxes and cropped that part of the image for classification by the CNN.

### 3.4 Explanation of used data sets

The data we have used comes from two different data sets, GTSRB and GTSDb. Both of which contains images of German traffic signs, where the difference is that GTSRB only contains cropped images of the actual signs whereas the GTSDb set contains images of the entire road and surroundings.

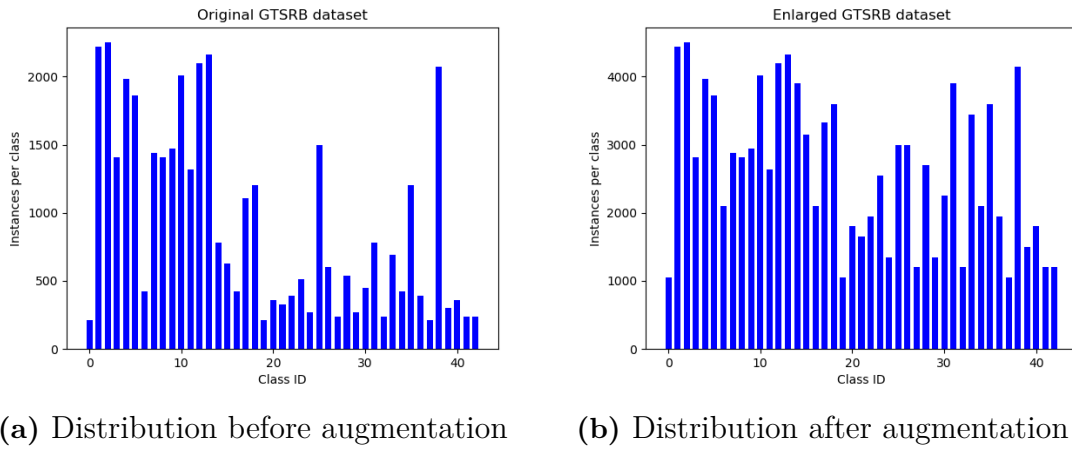
<sup>3</sup>Github repository for the Object Detection API: [https://github.com/tensorflow/models/tree/master/research/object\\_detection](https://github.com/tensorflow/models/tree/master/research/object_detection) (Available May 14 2018)

<sup>4</sup>The model can be downloaded from Tensorflow detection model zoo. [https://github.com/tensorflow/models/blob/master/research/object\\_detection/g3doc/detection\\_model\\_zoo.md](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md) (available May 14 2018)

### 3.4.1 The GTSRB data set for classification

GTSRB contains images of road signs belonging to 43 different classes. The set is divided into a training set, containing 39209 images and a test set containing 12630 images. The image size is ranging from 15x15 pixels to 250x250 pixels, and to each picture coordinates on the exact location of the sign is provided [51].

The classes of the GTSRB data set are unevenly distributed. A histogram of the occurrence of the different classes is shown in figure 3.2a [51].



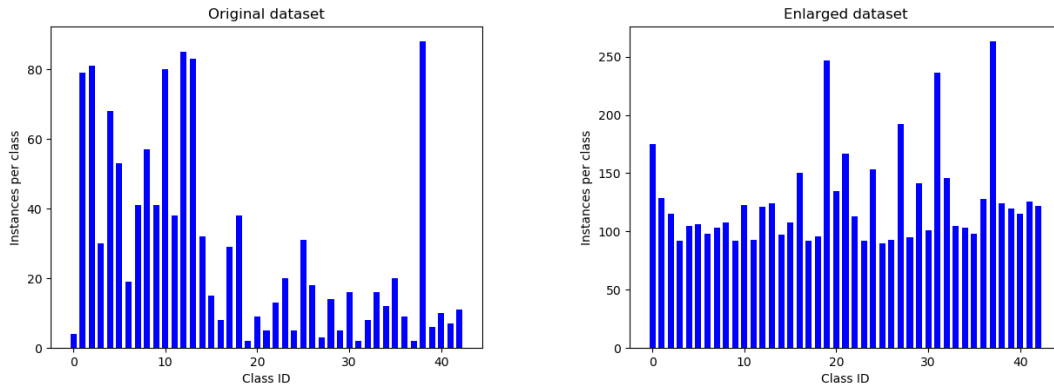
**Figure 3.2:** Figure 3.2a shows the histogram of the classes for the original training data attained from GTSRB webpage. Figure 3.2b shows the histogram of the classes after augmentation where all elements of a corner in the RGB matrices were set to zero. More images were created if the class was underrepresented, added to the original data-set.

An uneven data set could lead to biased results in the network, since predicting a often recurrent class will yield a better result no average than a class that is underrepresented. To smooth the distribution of the different classes we implemented a function that set all elements of the RGB matrices in a corner that is a quarter of the picture to zero. This operation makes that one quarter of the image black. For classes that are underrepresented (below 800 images) four images were created, each picture with a different corner where the pixels were set to zero. If the class had an acceptable number of images (between 800 and 1300), two new images were created, one with the upper left corner and one with bottom right corner blackened. Lastly, the category with too many classes (more than 1300 original pictures) got one random corner blackened. This implementation increased the training data set from 39209 to 113425 images. This smoothed the distribution as seen in 3.2b and was the extended data set used in the CNN.

### 3.4.2 The GTSDb data set for object detection

GTSDb contains 900 pictures, divided into 600 training images and 300 test images. Each picture is 1360x800 pixels in size and contains between 0 and 6 traffic signs, all of which belongs to the same classes as GTSRB. As the name reveals the focus on this data set is to detect where in the image a sign is located. The original focus on this data set was only to be able to detect the signs and not classify them [52]. For each picture in the data set there are information of the signs position given as a 4-tuple (position upper left corner, position lower right corner) and to which class the sign belongs. The images in the data set also includes traffic signs belonging to classes not included in the set and our algorithm will therefore not be trained to detect them [52].

As with the GTSRB data set the GTSDb data set is also unevenly distributed, with regard to classes, as can be seen in the histogram of figure 3.3a [52].



(a) Distribution before augmentation      (b) Distribution after augmentation

**Figure 3.3:** Histogram of the classes occurring in the used data sets for object detection. Figure 3.3a shows the distribution of the original GTSDb data set and figure 3.3b shows the distribution after augmentation.

In order to smoothen the distribution of the classes of the GTSDb data set we decided to create new images containing traffic signs. Each picture including the same information as the original images i.e. a 4-tuple with the position of each traffic sign in the picture and also which class the traffic sign belongs to. A pool of 25 background pictures none of whom contains any traffic signs was used to create the new images. One of these background are chosen randomly together with an integer 0 to 6, which decides how many traffic sign should be included in each picture. The classes of the chosen traffic signs were then randomly selected from a probability distribution designed to get an even final distribution of occurrences for all classes, in contrast to the very uneven distribution seen in 3.3a. More explicitly this distribution gave a probability of class  $n$  being chosen as  $P_n = \frac{1}{\sqrt{O_n}}$ , where  $O_n$  stands for the total number of occurrences of class  $n$  in the original data set. The images of traffic signs were all taken from the GTSRB data set. The traffic

signs were then randomly distributed in the background image. To prevent traffic signs to be positioned in the sky of the background image, this random position was only permitted to be chosen from a manually predefined accepted area. Lastly, the sharp edges between the traffic sign-crop and the background picture is smoothed by multiple Gaussian blurs. More explicitly a small Gaussian blur filter was used along the edges of the cropped traffic sign. This procedure was then done multiple times for incrementally bigger filter sizes, until the background and the edges of the cropped traffic sign at least visually seemed to blend together. We created 600 images and added to the original training data set, thus doubling the training set to 1200 pictures with a class distribution shown in figure 3.3b.

## 3.5 Software and setup

To implement the neural networks Python libraries Tensorflow and Keras were used. Tensorflow and Keras are specifically developed for implementing machine learning algorithms. A few additional Python libraries as well as MATLAB are used for preprocessing, augmentation and evaluating the results.

The CNN and Capsule Networks used for classification of the GTSRB images we trained on a computer with a GeForce GTX 1060 with 3 GB GDDR5 GPU, while the faster R-CNN is trained on Google Cloud's Machine Learning Engine<sup>5</sup>. When training on Google Cloud no GPU is used as CPU proved to be more cost efficient. Exact clock speed is not provided but the memory capacity is 15 GB. The main reasons to use Google Cloud when training the Faster R-CNN was that a graphic memory of 3 GB would have been insufficient.

---

<sup>5</sup>Webpage for Google's Cloud Machine Learning Engine: <https://cloud.google.com/ml-engine/> (available May 14 2018)

# 4

## Results

In this section the results from the three different networks, CNN, Capsule Network and Faster R-CNN are presented. The best accuracy on the test set for the CNN was 98.61 % and 99.62 % for the Capsule Network. The Faster R-CNN proved to be good at finding signs while struggling with classification. The combined classifier improved the performance compared to the pure Faster R-CNN.

### 4.1 CNN on par with human accuracy

Upon training the eight different models in table 3.1 to find the best architecture, every model managed to achieve more than 98% accuracy on the validation set for at least one of the three batch sizes used, see table 4.1. Network 4 achieved the highest validation accuracy on all three tests.

**Table 4.1:** The validation accuracy of eight different network structures, see table 3.1, after training using different batch sizes. All other parameters are fixed.

Network	5	10	20
1	0.9487	0.9886	0.9751
2	0.9870	0.9770	0.9739
3	0.9910	0.9801	0.9862
<b>4</b>	<b>0.9935</b>	<b>0.9901</b>	<b>0.9867</b>
5	0.9837	0.9860	0.9828
6	0.9910	0.9872	0.9625
7	0.9861	0.9704	0.9707
8	0.9921	0.8977	0.9807

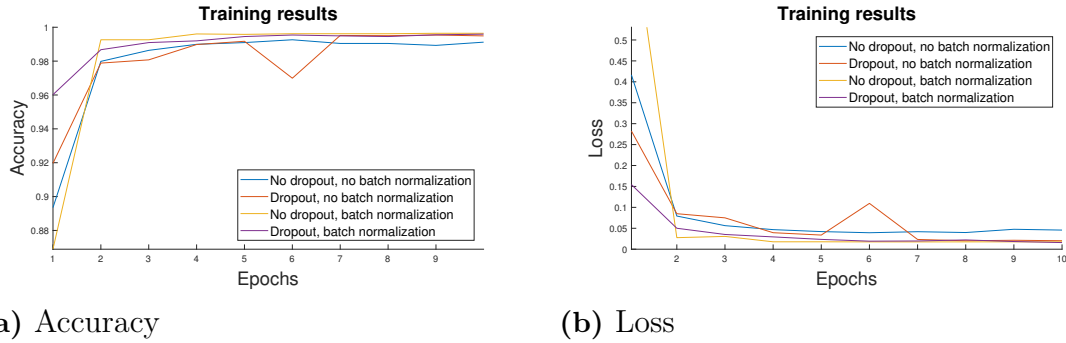
Observations from tests using different regularization techniques show mainly that dropout works best on the fully connected layers. The three best performing networks for each dropout rate do not implement dropout on the convolutional layers, see table 4.2. The results also show that batch normalization provides good results when applied to the conv layers.

#### 4. Results

**Table 4.2:** Different use of regularization on network 4 from table 3.1 ranked by best performance on the validation set with a batch size of 15. The interpretation being: the architecture for the best performing network with a dropout rate of 20 % is when dropout is used on the fully connected layers together with batch normalization on the convolutional layers.

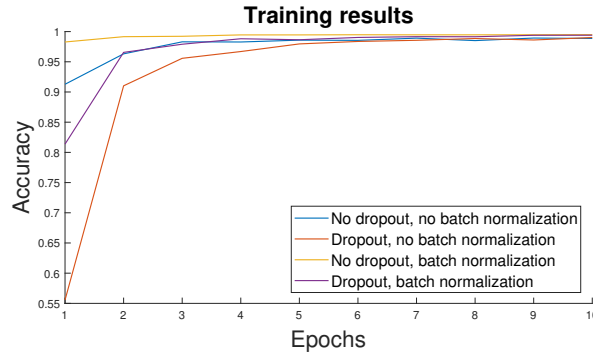
Ranking	Dropout rate	20 %	40 %	60 %
1	Dropout Batch norm.	Fully connected Convolutional	None Convolutional	None Convolutional
2	Dropout Batch norm.	None Convolutional	Fully connected Convolutional	Fully connected Convolutional
3	Dropout Batch norm.	None None	Fully connected Everywhere	None None

Further testing on the four selected architectures, with more restrictions on how regularization is used, show that any of the allowed combination of these techniques perform better than the network without any regularization. This is easiest seen in figure 4.1b and 4.1a, which shows the loss function and validation accuracy of the network after each epoch of training. There is a distinct gap between the network without any regularization, represented by the blue line, and the other three networks. This gap can not be seen as clearly in figure 4.2 but it is noticeable that the dropout takes more time to train compared to other network without dropout and that batch normalization speeds up training.



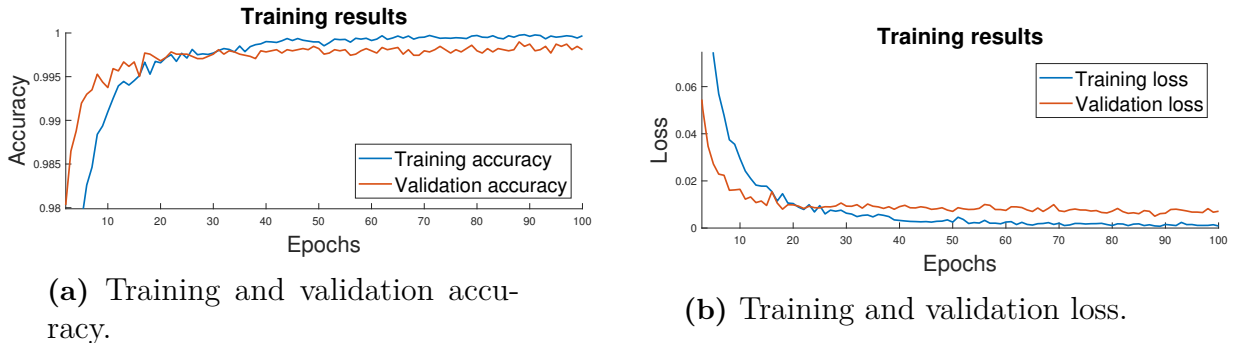
**Figure 4.1:** Graph showing the validation accuracy and loss of the four different models after each epoch of training. The networks were trained with batch size 15, a learn rate of 0.02 and dropout rate of 40%.

The training process of the selected network with batch normalization on the convolutional layers and dropout on the fully connected layers can be seen in figures 4.3a and 4.3b. The results when evaluated on the test set can be found in table 4.3. The confusion matrix containing the predictions of the network is visualized in figure 4.4. The network had problems accurately classifying class 27, Caution Pedestrian, which it classified as class 1, Speed Limit 30, almost half the time. Training the same network on the extended data set increased its performance. After increasing



**Figure 4.2:** Graph showing the validation accuracy of the four different models after each epoch of training. The networks were trained with batch size 15, a learning rate of 0.01 and a dropout rate of 50%.

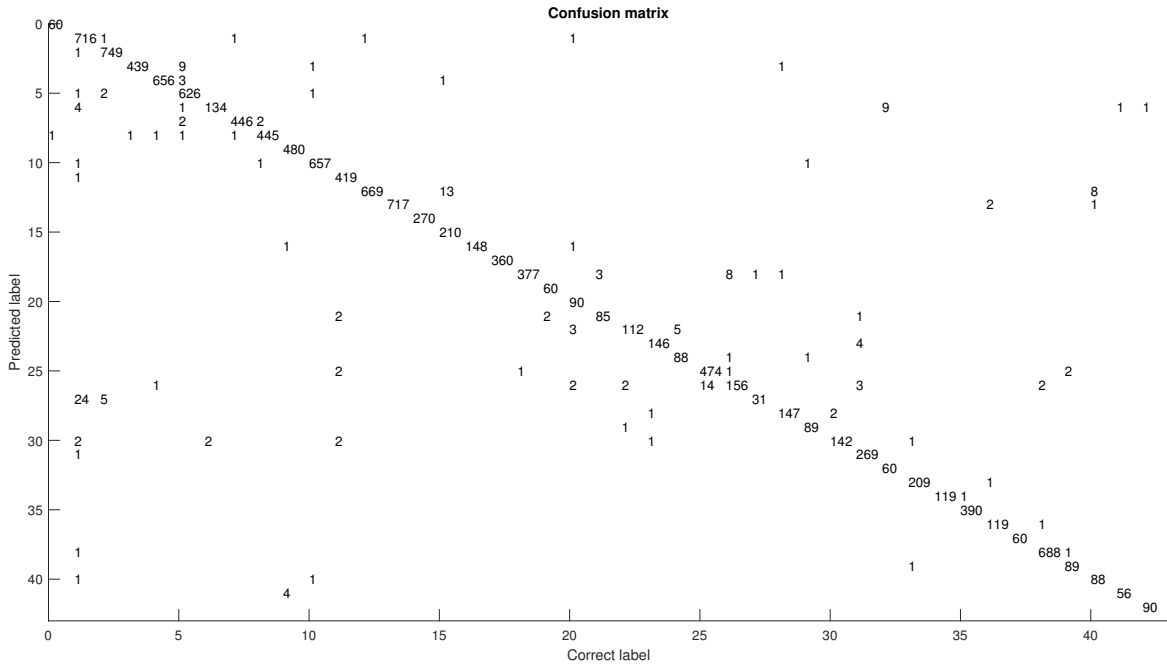
the size of the network and once again training it on the extended data set the test accuracy was slightly increased, see third row in table 4.3. The final network now classifies class 27 correctly, as can be seen in appendix B figure B.1. A comparison between the training and validation results, see figures 4.5a and 4.5b, show insignificant levels of overfitting. Performing seven additional separate trainings of the final network showed that the model consistently achieves similar scores. This average score is however slightly lower than that of the first iteration, see fourth row in table 4.3.



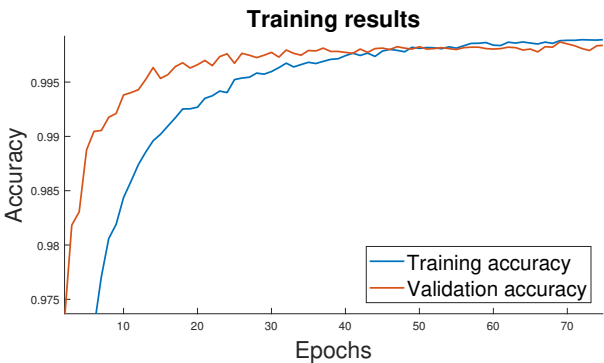
**Figure 4.3:** Training and validation accuracy and loss of the regularized network. The network was trained with batch size 15, a learn rate of 0.005 and dropout rate of 37%.

**Table 4.3:** Test accuracy and loss of the regularized and the final network. The average result of an ensemble of eight retrained instances of the final networks is also shown. The networks were trained with batch size 15, a learn rate of 0.005 and dropout rate of 37%.

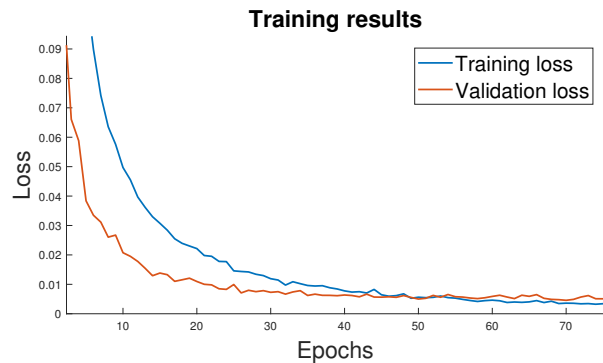
Network	Data set	Epochs	Test Accuracy	Test Loss
Regularized	Original	100	98.14 %	0.0891
Regularized	Extended	50	98.58 %	0.0616
Final	Extended	75	98.61 %	0.0763
Final (average of 8 networks)	Extended	75	98.49 %	0.0962



**Figure 4.4:** Confusion matrix of the regularized network. Each row represents the predicted class and the columns indicate the true label. The network was trained with batch size 15, a learn rate of 0.005 and dropout rate of 37% for 100 epochs.



(a) The training and validation accuracy.

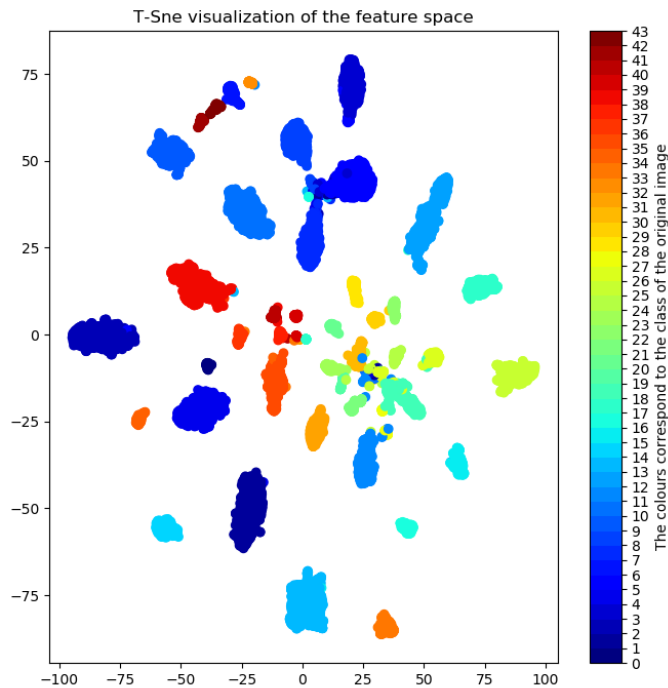


(b) The training and validation loss.

**Figure 4.5:** Training and validation loss and accuracy of the final model. The network was trained with batch size 15, a learn rate of 0.005 and dropout rate of 37%.



Besides accuracy and loss, it is of interest to analyze what the network is learning to conclude if it is robust. Figure 4.6 shows the 1024-dimensional vectors of the last dense layers, after PCA and t-sne was applied. The figure seems to show evidence that the network is performing clustering by class in this layer. Supporting evidence of this is given in table 4.4.



**Figure 4.6:** Visualization of the 1024-dimensional dense layer vectors corresponding to every image in the test set, color coded according to the image class. The dimensions were reduced using PCA and t-sne. The scatter plot suggests that the network manages to cluster the traffic signs according to their classes, a fact that could help explain why the prediction accuracy is relatively high.

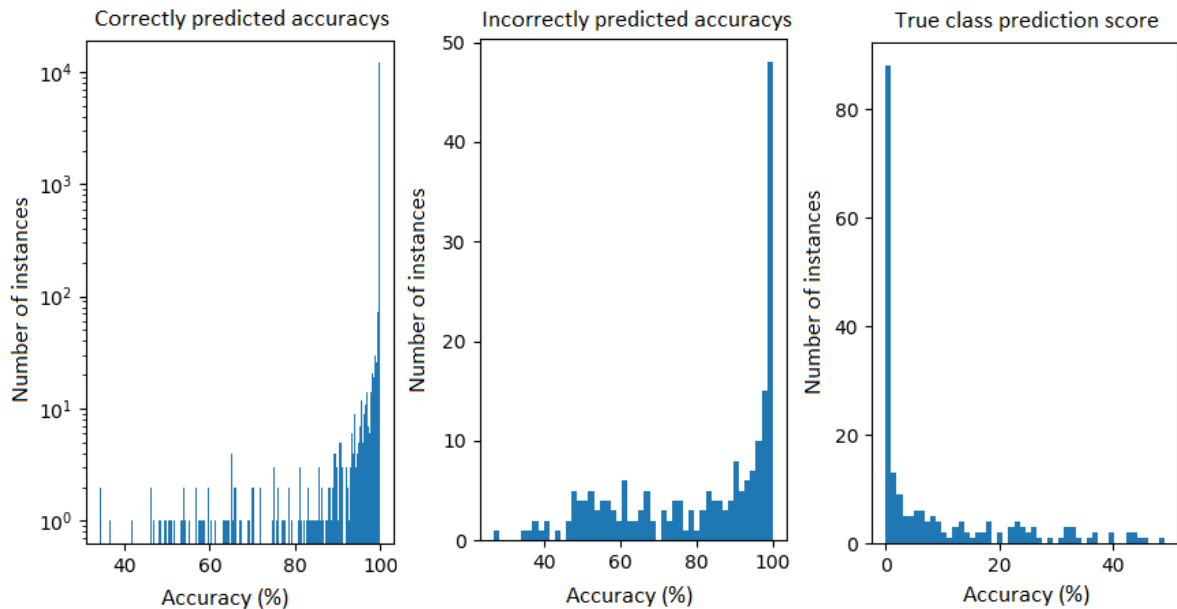
**Table 4.4:** K-Nearest neighbor prediction accuracy in the 1024-dimensional feature space after the last dense layer, using all images in the GTSRB test set. The results supports the class clustering hypothesis, see figure 4.6.

	Accuracy, L1-norm	Accuracy, L2-norm
<b>1-Nearest neighbor</b>	99.64%	99.63%
<b>5-Nearest neighbor</b>	99.48%	99.45%
<b>10-Nearest neighbor</b>	99.04%	98.92%

Another important result is to look at how certain the network is when predicting classes. As can be see in figure 4.7 the network is often close to 100% certain of its choice, including the times where it incorrectly predicted a class. Another metric that gives some sense on how close the network was to predict the right class in cases where it failed is top-X accuracy, see table 4.5. The results in the table suggests that the network manages to find important information about the true class even for misclassified images.

**Table 4.5:** Top-X test accuracy for X=2, 3, 4 and 5. At least for some instances it seems that the network is close to predicting correctly, even though it misclassified the image.

	Test accuracy
<b>Top-2</b>	99.29 %
<b>Top-3</b>	99.57 %
<b>Top-4</b>	99.66 %
<b>Top-5</b>	99.74 %



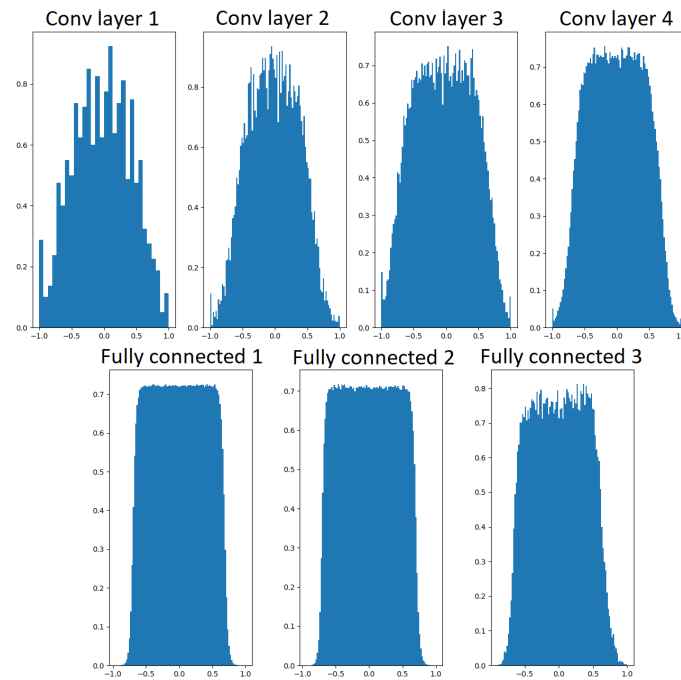
(a) Prediction scores for each of the correctly classified images.

(b) Prediction scores for each of the misclassified images.

(c) Prediction scores on true class for each of the misclassified images.

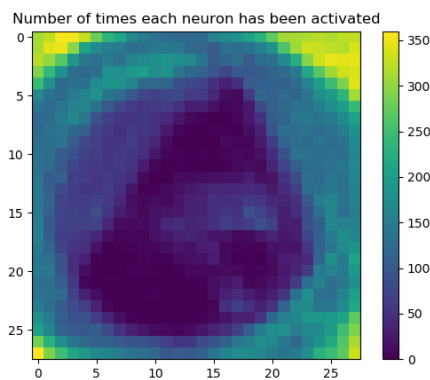
**Figure 4.7:** Graph (a) shows the predicted class score for every correctly classified image (observe the logarithmic scale). Almost every correct classification is done with close to 100% certainty. Graph (b) shows the incorrectly classified images. When the network misclassifies an image, it does it with a slightly lower certainty between 50-100%. Graph (c) shows the prediction score for the correct class for every misclassified image. It seems that the network is certain in its predictions, no matter if it ends up being correct or incorrect.

The distribution of the weights in each layer is shown in figure 4.8. Due to batch normalization the weights in the network should not grow large, which the figure also suggests is the case. All weights seems to follow a Gaussian distribution.

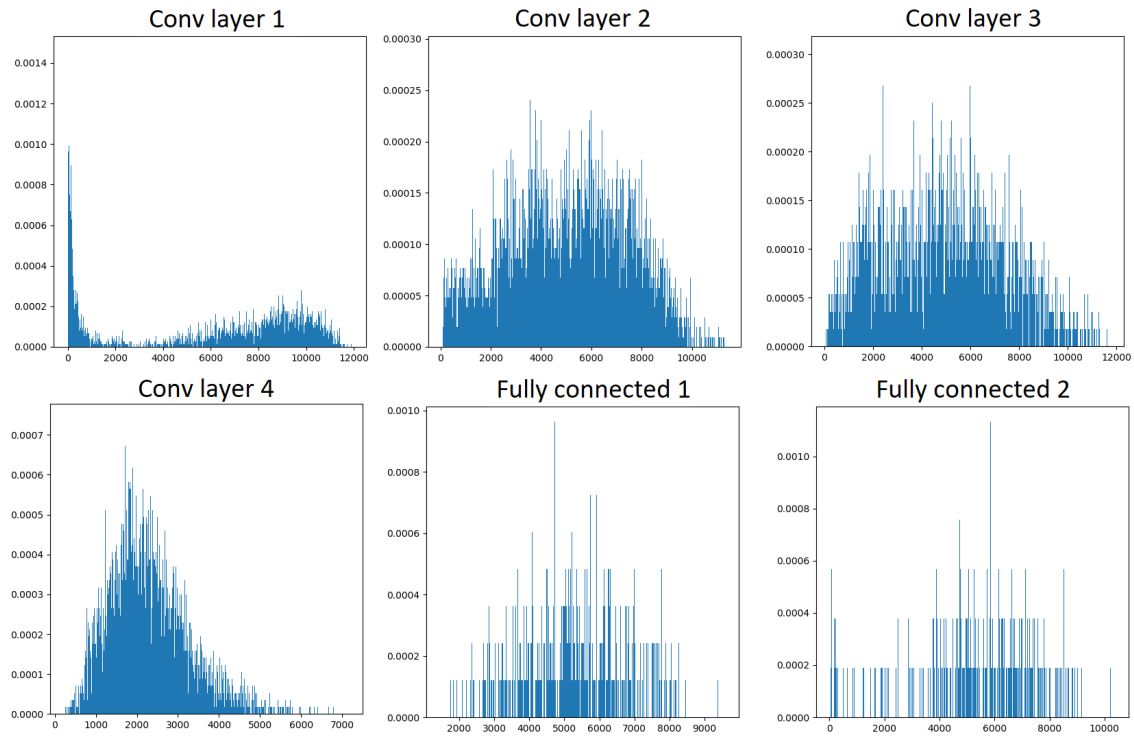


**Figure 4.8:** The normalized weight distribution for each layer in the CNN. Each layer's normalization is calculated independent from the next. The figure seems to suggest that all weights follow a Gaussian distribution.

Striving for an efficient network includes that all neurons should be active at least once, with the definition of active as any number larger than 0. The distribution of how often neurons in layers are active is shown in figure 4.10. A further investigation of the dead neuron phenomena shows that there are only 17 out of 68736 neurons in the model that never activate for any of the 12630 test images - less than 0.025% - and every neuron is inactive at least once. All the inactive neurons arise from the same filter in the first conv layer, see figure 4.9.



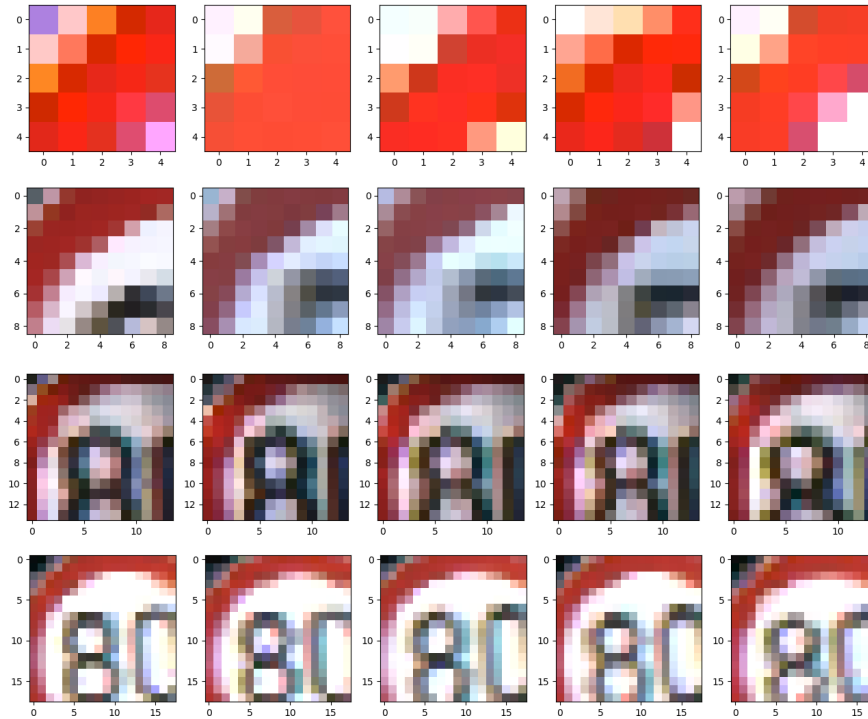
**Figure 4.9:** Number of times each neuron was activated given all 12630 images, shown for the activation map containing all 17 dead neurons.



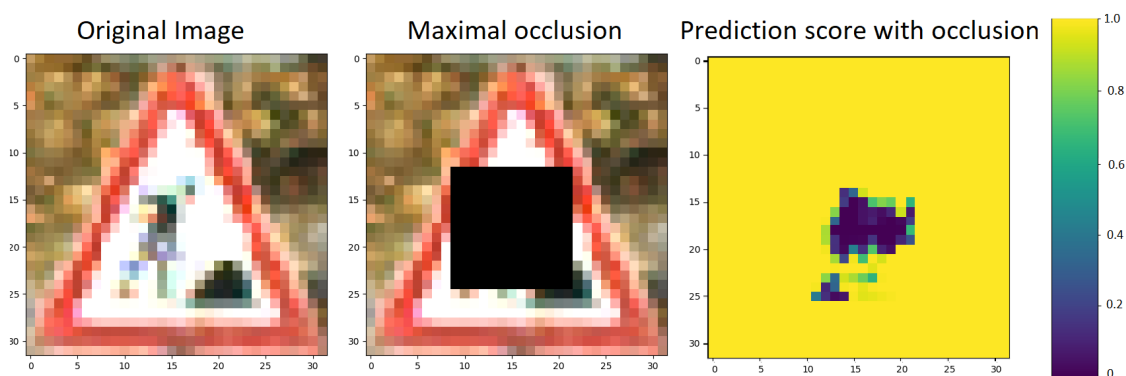
**Figure 4.10:** Histogram of how often the neurons are active when feeding the test set of 12630 images. Active is defined as any number larger than 0. In all layers except the first conv layer most of the neurons are active between 1000 and 10000 times. In the first conv layer the majority of neurons are active less than 1000 times. The only neurons that never activate, and thus are considered dead, all lie in conv layer 1.

By looking at a specific neuron’s receptive field for those images that maximizes its output, one can see what features activates the neuron. In figure 4.11 five neurons’ receptive fields in four different layers of the CNN are shown. It is evident that the given neuron activates for images that contain similar information, which strengthens the conclusion that the convolutional filters are looking for specific features. It is also clear that filters deeper in the network are looking for bigger structures (which is implied by the fact that the receptive field enlarges) with more complex features, which is what the theory predicts.

It is also possible to learn what explicit features in an image are crucial for classification by iteratively occluding parts of the image and trying to classify the image after each occlusion, an example from this procedure is shown in figure 4.12. The heat map on the right hand side of the image correspond to the prediction score as a function of what part of the image was occluded. The occlusion tests consistently showed that distinct parts of the traffic signs are crucial for classification.



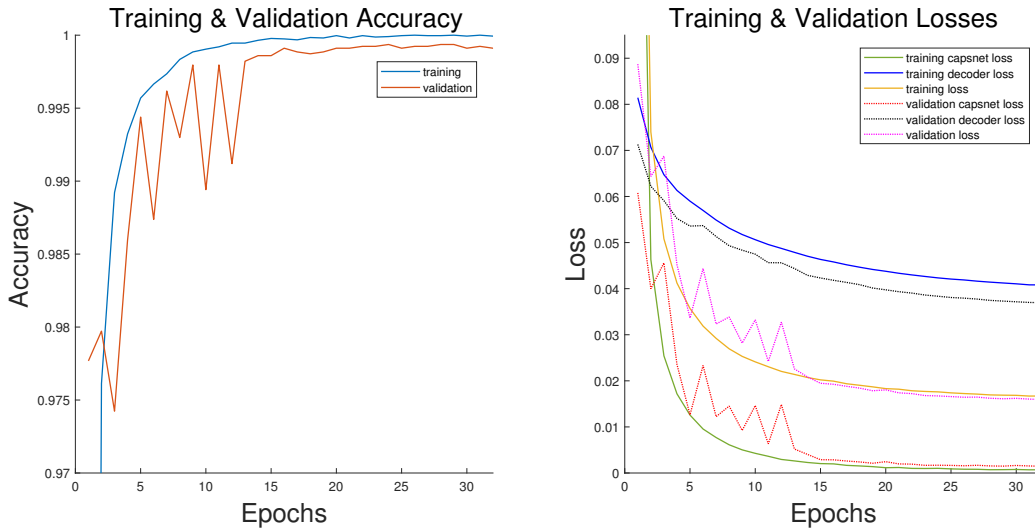
**Figure 4.11:** The five patches of an image within a certain neurons receptive field that maximally activated said neuron. The top row corresponds to a neuron in the activation map after the first convolutional layer, the second row to the second convolutional layer and so on down to the fourth layer. Each column represents a different picture.



**Figure 4.12:** A representative example from the occlusion tests performed to investigate which features are crucial for classification of images. The figure seems to suggest that the image gets misclassified as soon as the road worker is occluded. If only the background, or parts of the traffic sign's edge, is covered then the prediction score does not change at all.

## 4.2 Capsule Network state of the art performance

The result using the initial implementation explained in 3.2 was a training accuracy of 99.99%, validation accuracy of 99.86% and a test accuracy of 98.47%. The result after each epoch of training can be seen in figure 4.13. This was with a 20/80 validation/training split. Since this already gave a training accuracy close to 100% it was concluded that the architecture of the network was suitable since it could be seen that it gave the network the ability to learn the relevant features.



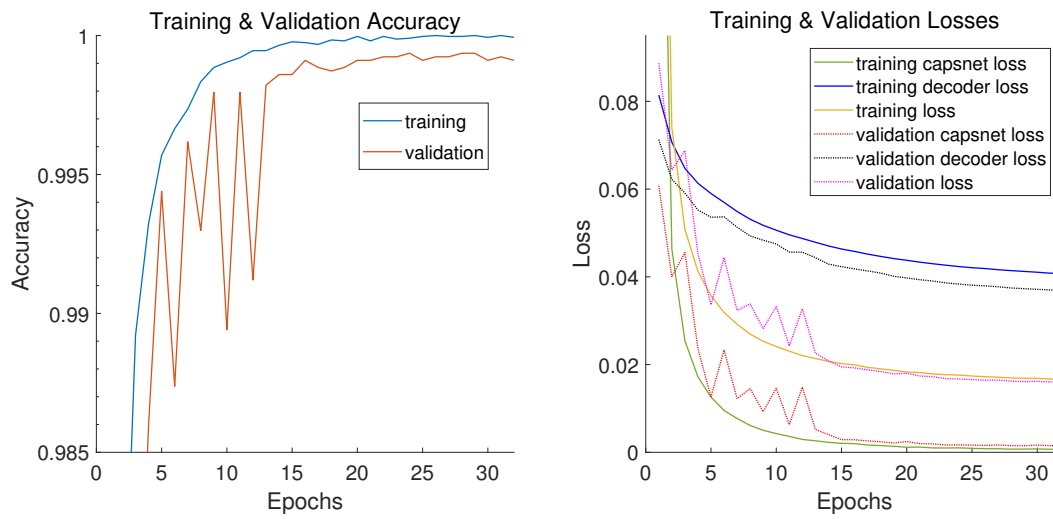
**Figure 4.13:** Accuracy and loss after each epoch of training for the initial implementation. This gave a final training accuracy of 99.99%, validation accuracy of 99.86% and test accuracy of 98.47%.

Different optimization algorithms were tested and RMSprop achieved the best validation accuracy as can be seen in table 4.6. Then batch normalization was used after the first convolutional layer and the two first fully connected layers in the decoder which gave the result seen in figure 4.14.

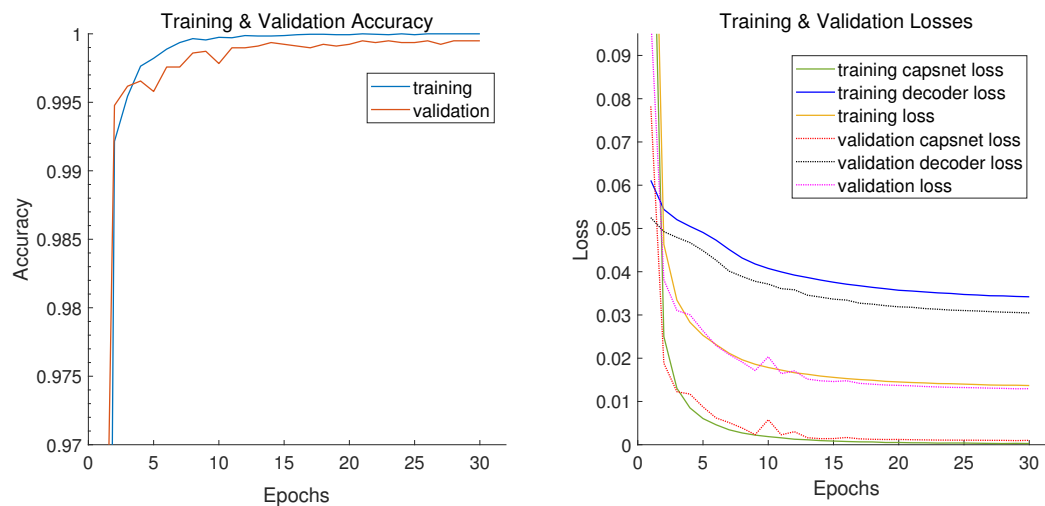
**Table 4.6:** Comparing different optimization algorithms for Capsule Network. RMSprop gave the best performance, shown in bold.

	Training Accuracy	Validation Accuracy
<b>Adam</b>	99.99%	99.86%
<b>Nadam</b>	99.97%	99.83%
<b>Adagrad</b>	90.25%	95.23%
<b>RMSprop</b>	<b>99.99%</b>	<b>99.92%</b>

After implementing improved contrast normalization a test accuracy of 99.35% was achieved, displayed in figure 4.15. The SELU activation function lead to a reduced test accuracy while the ELU activation function lead to a small increase to a test accuracy of 99.36%.



**Figure 4.14:** Accuracy and loss after each epoch of training when using RMSprop and batch normalization. This gave a final training accuracy of 100%, a validation accuracy of 99.94% and a test accuracy of 99.02%



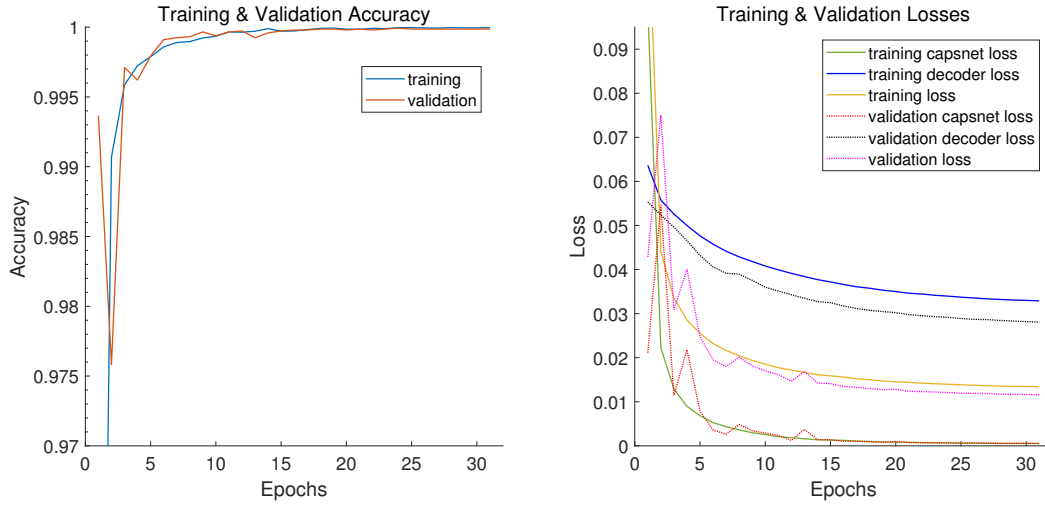
**Figure 4.15:** Accuracy and loss after each epoch of training after applying improved contrast normalization. This gave a final training accuracy of 100%, a validation accuracy of 99.98% and a test accuracy of 99.35%

#### 4. Results

By performing data augmentation and using an ensemble method, where the average of networks with individual F1-scores of 99.33%, 99.35% and 99.52% was taken, it was possible to reach a final F1-score and test accuracy of 99.62%, which means that only about 50 of the 12630 test images were wrongly classified, with the accuracy for many of the classes being 100%. Table 4.7 and figure 4.16 displays the result.

**Table 4.7:** Accuracies with different augmentation and with an ensemble method.

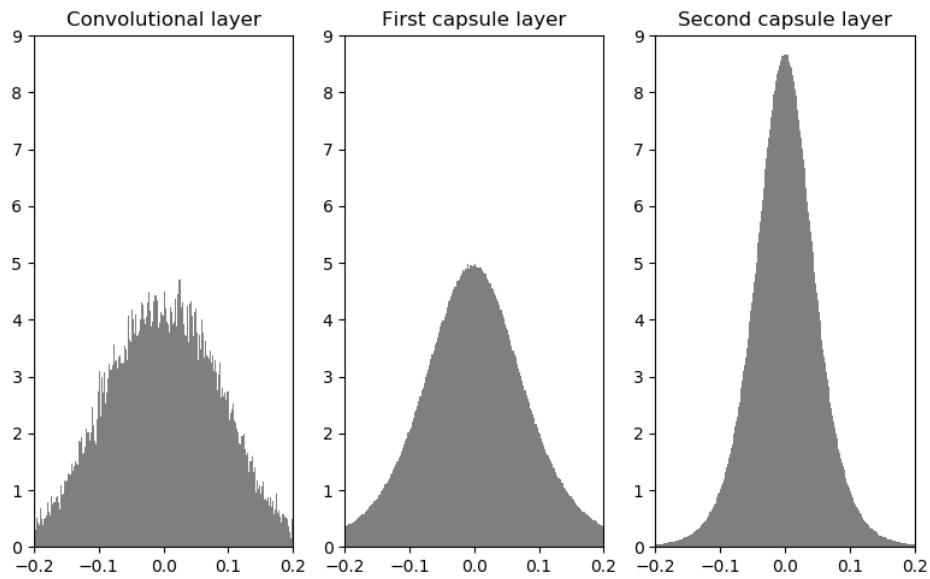
	Training Acc.	Validation Acc.	Test Acc.
<b>Augmentation 1</b>	100%	99.99%	99.33%
<b>Augmentation 2</b>	100%	99.99%	99.35%
<b>Augmentation 3</b>	100%	99.99%	99.52%
<b>Ensemble method</b>	100%	99.99%	99.62%



**Figure 4.16:** Accuracy and loss after each epoch of training for the network that achieved highest accuracy without using an ensemble method. This gave a final training accuracy of 100%, a validation accuracy of 99.99% and a test accuracy of 99.52%

For the version that gave a test accuracy of 99.52% of our Capsule Network, the weights in the different layers were distributed as shown in figure 4.17. The result of the reconstruction part of the network is shown in figure 4.18, where fifty traffic signs were run through the decoder. The figure provides some evidence for that the network has learnt to synthesise the most important class features in the DigitCaps, since the decoder manages to recreate recognizable images. Lastly, another thing to analyze in the Capsule Network is what each dimension of a capsule output represents. To do this for the capsules in the DigitCaps layer, the decoder is fed with a vector that is shifted one dimension at a time. The reconstructed images will then show what the different dimensions encode. Figure 4.19 shows the result of the manipulated vectors. The result seems to suggest that different dimensions control e.g. contrast, sharpness and brightness. The confusion matrix for the network with test accuracy of 99.62% is displayed in figure 4.20.

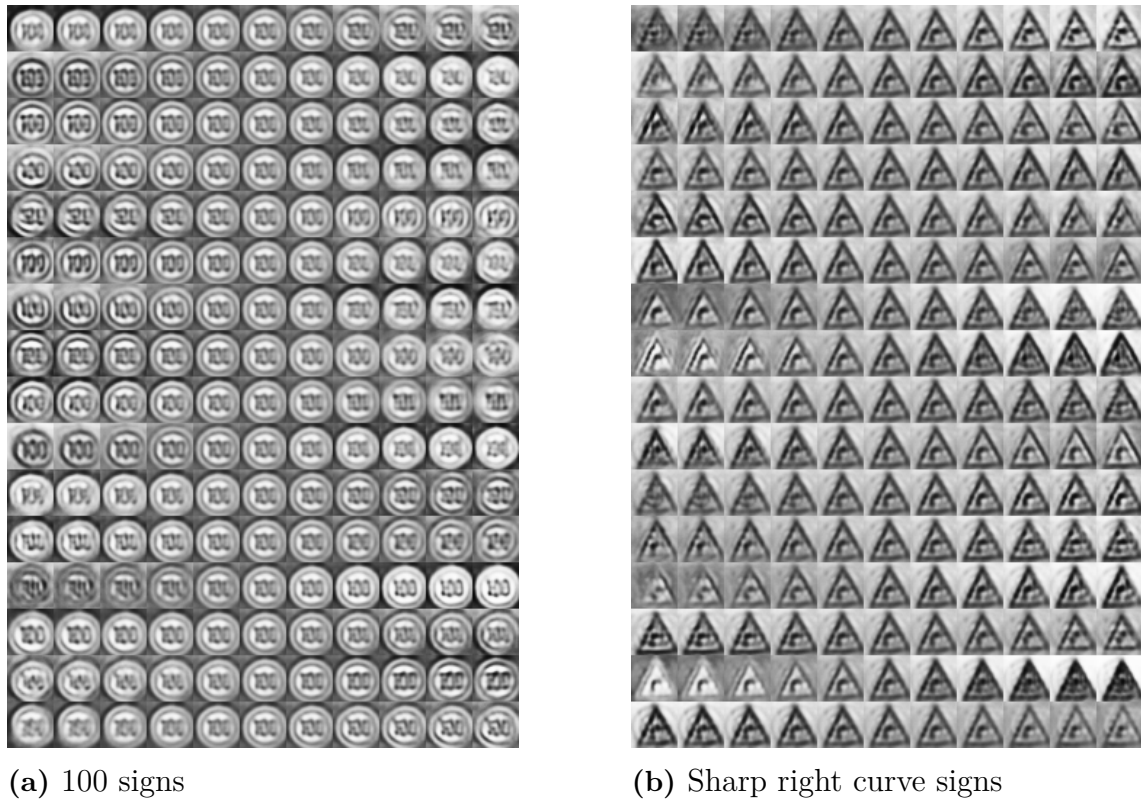




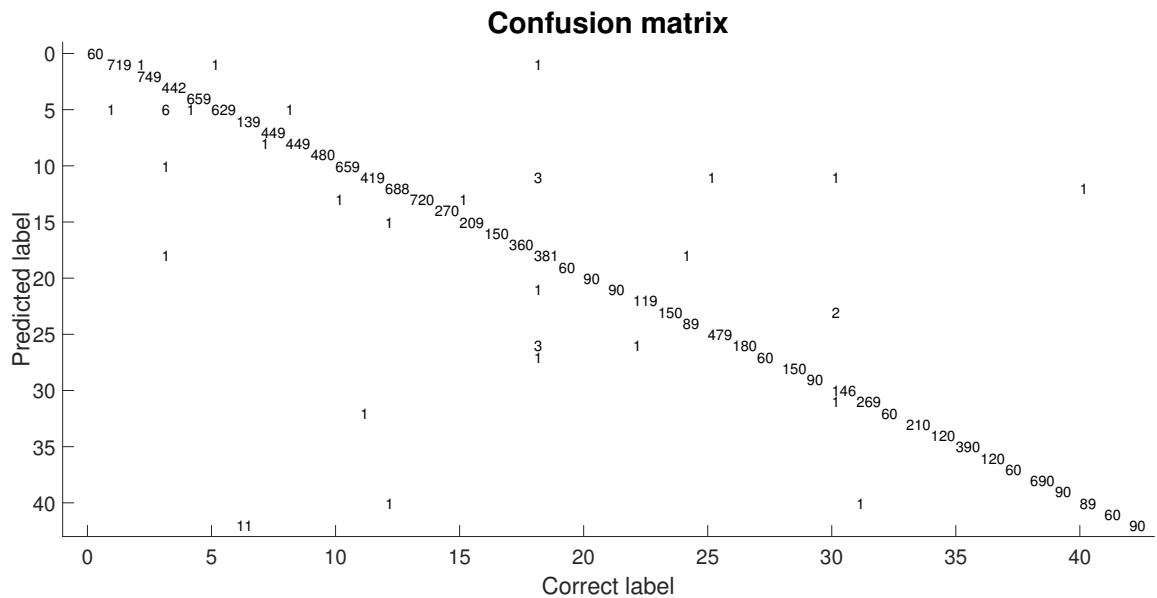
**Figure 4.17:** Weight distribution of the first 3 layers of the capsule network.



**Figure 4.18:** Input traffic signs and the corresponding reconstructed images by the decoder network. The real are on the upper half of the image and the reconstructed on the lower half.



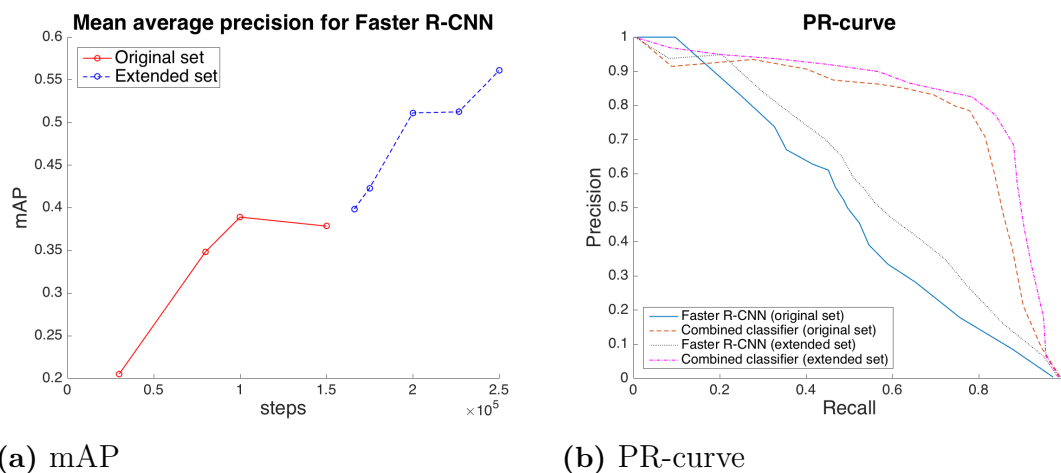
**Figure 4.19:** Manipulated images created by the decoder part of the network. The images are created by manipulating each of the 16 dimensions in the input vector to the decoder part of the network. One dimension is shifted by ten different values ranging from -0.25 to 0.25. The original reconstruction is in the middle and the negative shift is on the left and the positive are on the right.



**Figure 4.20:** Confusion matrix for the final version of Capsule network that gave 99.62% test accuracy. Each row represents the predicted class and the columns indicate the true label.

### 4.3 Faster R-CNN shows promising detection but classification proved harder

The most important measurement of the performance of the Faster R-CNN is mean average precision (mAP) and figure 4.21a shows mAP as a function of training steps. When training on the original GTSDb training set the mAP increased continuously up to 100000 steps and decreased slightly at 150000 steps. Given that this corresponds to a 50 % increase of training this indicates that the network is close to its maximum performance. After 150000 steps training was continued on the extended set and mAP increased once again and showed no sign of flattening when training was terminated at 250000 steps with a mAP of 0.56.



**Figure 4.21:** (a) mAP values for Faster R-CNN trained on the original and extended GTSDb training set. (b) Precision-Recall (PR) curves for faster R-CNN trained on the original and extended GTSDb training set respectively. PR curves for two combined networks that use our GTSRB CNN for classification are also displayed in the figure.

To illustrate what mistakes the network is making three example output are shown in figures 4.22 and 4.23. To generate this output, a confidence score threshold of 0.5 was used and the maximum numbers of predictions included for each input picture is limited to 20. As can be seen in figure 4.22, the network makes all three types of possible errors: incorrectly predicts the presence of a sign where there is none, fails to find signs in the picture and predicts the wrong class when it finds a correct bounding box. When going through a larger sample of outputs it is clear that the network finds most of the signs and the dominating issue is that the predicted class is incorrect. It is also evident that the network performs much better on some classes, for instance Priority road and Give way, than it does on others, such as the different speed signs. It should also be noted that this corresponds in some extent to the distribution of the training set. The same pattern is repeated when going through the output after training on the extended set, exemplified in figure 4.23. The main issue is still that the network finds most signs but assigns incorrect classes. However,

given the increased mAP-score the issue is evidently smaller.

In order to compare the performance of Faster R-CNN with our combined classifier, Precision-Recall (PR) curves for both methods were created and are displayed in figure 4.21b. Ideally the PR-curve should enclose an area equal to one, which would correspond to the network never making any false prediction while simultaneously finding and classifying all signs in the test set correctly. As can be seen in the figure, precision as a function of recall is higher for the combined classifiers except for the lowest and highest recall levels. In total the curve for the combined classifiers encloses a significantly larger area than the corresponding Faster R-CNN. It is also worth noting that the curve for the Faster R-CNN trained on the extended set is higher as expected given the higher mAP. It is also clear from the figure that the combined classifier increased the performance to approximately the same extent, when based on the networks trained on the original and extended set respectively.



**Figure 4.22:** Example output from the test set after training the Faster R-CNN model for 150000 steps on the original training set. In the top image it can be seen that the network makes a correct prediction of the Priority road sign and it does find both 70-signs. However the speed signs are wrongly classified as 60 and 30. In the middle picture the network does not find the 50-sign. In the bottom picture the stop-sign and keep right-sign are both predicted correctly, however in the left part of the picture the presence of a sign is wrongly predicted.





**Figure 4.23:** Example output from the test set after training the Faster R-CNN model on the extended training set. In the top image the model incorrectly predicts the Volkswagen-logo as a sign. In the middle image the Priority road sign is correct, but the network classifies the 70-sign as a 50-sign. In the bottom image the network fails to find and classify the left stop sign.

# 5

## Discussion

In this section the results are discussed. We set out to implement neural networks that should perform close to the best listed accuracies for GTSRB and both networks achieved good results. We also had an aim to explore object detection and we managed to implement a working detection network, using a Faster R-CNN.

### 5.1 The simple CNN structure is able to find complex features

The process of improving our network started with altering the structure and tuning the hyperparameters. Upon reaching a satisfactory model we implemented data augmentation as described in section 3.4.1. As can be seen in table 4.3, training on an extended data set significantly reduced the test error. The gap between the training and validation curves presented in figure 4.3 suggests slight overfitting on the training set. When training the final model using the extended set this gap greatly reduced, see figure 4.5. A conclusion that can be made is that by increasing the amount of data we managed to reduce overfitting, which consequently led to higher general performance of our model.

The way we augmented the training set might not have been the most efficient. By trying out different methods the results might have been improved further. The augmentation we used, in which we removed one corner of the picture, might train the network to find sharp edges that are non-existent in the test set. Other data augmentation that could have been used is flipping some signs that are symmetrical, rotate the picture or use some whitening transformation. To analyze the network more in depth we visualized every misclassified image, together with its predicted class and confidence level. 25 out of the misclassified images are illustrated in figure B.2. The first observation that can be seen is that most of the misclassified images seems to have a very high or a very low contrast. This implies that we would probably be able to increase our test accuracy if we had used contrast normalization as a preprocessing method. It is also evident that many of the misclassified images are very alike, for example the three Priority road signs illustrated in figure B.2. A

reasonable assumption thus is that of the misclassified images, it is only a fraction that correspond to distinct flaws - the majority is just more instances of the same issue.

Another important step in our networks architecture was the implementation of batch normalization, which reduced both the training time and the risk of the model diverging. The result of this is that the updates of weights remain small. A Gaussian distribution of the weight values, seen in figure 4.8, indicates that the network uses all of its parts. A distribution centered around a negative value would indicate that most of the neurons, since we are using the ReLU activation function, would be dead i.e. equal to 0. This is not the case for us, which is confirmed when we look at how often the neurons were active when feeding the network the 12630 images from the test set, see figure 4.10. From the distribution it is clear that our network does not suffer from a dead neuron issue, since most of the neurons are active for about half of the inputs and only 0.025% are dead. All dead neurons are all caused by the same filter in the first convolutional layer, see figure 4.9. In this case the filter seems to look for features that never exist in the middle of any traffic sign, but relatively often in the corners - which suggests that it might be looking for round edges or other peripheral details. No neuron in the feature map was active for more than 3% of the input images, suggesting that the searched feature is rather rare as well. Lastly, another interesting phenomena is that a majority of the neurons in the first conv layer only activate a few times, as can be seen in figure 4.10. According to the theory this first layer should be searching for sixteen low level features - one feature for each filter - such as straight lines or curves of different colors, a statement that is supported by looking at figure 4.11. An explanation to the distribution for the first layer is thus that these sixteen low level features only cover a small part of the input images. This is not surprising, since a traffic sign consist of a lot more than just - for example - a red surrounding circle. Taking this one step further it might even be possible to conclude from the distribution for the first layer that only a small portion of the pixel information in the traffic signs are relevant for classification. This statement is supported by the observations from the occlusion test, see figure 4.12, which will be discussed further in a later paragraph.

The final network is on par with human performance, 98.84% [7], which is strongly verified when examining the misclassified images. It is evident that the average human eye would have a hard time doing a better job than our classifier on most of those images, since many of them are severely blurred. However, some images are very easily classified manually but missed by our network. Once again the three Priority road signs can be used as an example, see figure B.2. In these cases we can suspect that the classifier has a hard time due to that parts of the sign is blocked or partly shaded. This is supported by the occlusion test performed on a correctly classified Priority road sign, which clearly showed that the most important feature for classification of this class is the big yellow patch. This fact in itself is not surprising because only a few classes have any yellow features at all which means that this corresponds to a distinct trait for the Priority road class.



It is also interesting to analyze the distribution of the networks prediction score, as seen in figure 4.7. From it we can clearly see that every time the network correctly classifies an image it does so with a very high (close to 100%) certainty. To some extent this seems to be the case for misclassified images as well. However, we can clearly see that there are instances where a misclassification was accompanied with a low certainty score, sometimes as low as 20%. This could mean that the network in those cases might be almost equally certain on another - possibly correct - class. However, the rightmost graph in figure 4.7 clearly shows that this in fact is not the case - since for most of the misclassified images the network gave a very low (close to 0%) prediction score for the correct class. In short: The classifier almost every time claims to be very confident, no matter if it is correct or incorrect. With that said there are some instances that were incorrectly classified but nevertheless got a high score for the correct class - the most extreme case gave a prediction score of 49% on the correct class and 51% on the incorrect class. Another indication that the network might have learnt and found important features even in cases where it ended up misclassifying an image is given by table 4.5. The fact that the top-3 accuracy score is as high as 99.57% suggest that the network is not too far from performing on par with the state-of-the-art networks.

To get a understanding for what the network is learning at the last dense layer we can analyze the meaning of figure 4.6 and table 4.4. The strong evidence of clustering given in the figure suggests that the layer is synthesizing core characteristics for each class, which is not an unrealistic conclusion since it is this vector that precedes classification. Further evidence that clustering in this layer is crucial for classification can be found by interpreting what the last weight matrix does. This is because it is a linear matrix operation, which could in some sense be interpreted as dividing the 1024 dimensional room into 43 subspaces with hyperplanes. Bad clustering would thus mean that images corresponding to a certain class would end up in the wrong subspace which would lead to them being misclassified. Lastly, the high classification scores given in table 4.4 indicates that this clustering is not only present, but also effective.

To visualize what features the convolutional filters have learned to detect we used multiple approaches. The results were consistent and proved that the filters are indeed trained to look for very well-defined features, as shown in figure 4.11. It was also very consistently shown that deeper layers are looking for more complex structures, whilst the first convolutional filters are looking for mostly edges and straight lines of different colors. Figure 4.11 suggests for example that the first filter chosen in this illustration seems to be looking for red upward sloping lines, the second filter is looking for red arches and the third and fourth filter are looking for parts of white circles, enclosed by a red arch with black digits in them. The third filter seems to only look for the number 8, whilst the last layer searches for the number 80. Figure 4.12 further strenghtens the conclusion that the filters are extracting valuable information from features. In this case it is evident that the existance of the road worker seems to be the deciding factor in the classification of the image. It is also interesting that the prediction score does not lower if any other

parts of the traffic sign is occluded, suggesting that the network is robust against partial occlusion from shade or other objects. Overall the occlusion test gave results that implied that the network has learned to look for the same features as the human eye, such as the numbers on a speed sign or the explanatory object in the middle of a warning sign.

## 5.2 Capsule Network performs well outside of the simple MNIST data set


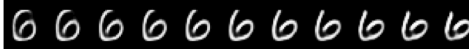



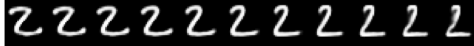
The main goal with our capsule network implementation was to reach an accuracy close to the current highest accuracy for GTSRB. This was accomplished by getting an test accuracy of 99.62% which is only 0.09% lower than the current highest accuracy which of May 13 2018 is 99.71% [8].

Another interesting result was how well the initial implementation performed, by using a network originally created for MNIST we were able to achieve a training accuracy of 99.99% and a validation accuracy of 99.86%. One reason for this is the similarities in the two data sets, many similar features can be found in both data sets, especially for the traffic signs which contain numbers. This also shows Capsule Networks ability to generalize and that the same network structure can give good results on different data sets. One reason for this might be that Capsule Networks take in regard both features and pose.

The first results had a gap of 0.13% between the training and validation accuracy, this shows that the network to a small extent overfitted to the training set. Since the network has a high number of learnable parameters, about 21 million, it is not unexpected that it is able to overfit to the training set. We assume that the main reasons that it does not overfit more to the training set is the decoder loss which gives the network a built in regularization and that the training is run for a suitable number of epochs. By finding a optimization algorithm that is better suited for our data set and by introducing additional regularization through batch normalization and data augmentation it was possible to remove the overfitting and get a validation accuracy that converged to approximately the same level as the test accuracy, as can be seen in figure 4.16. Figure 4.17 shows that the weights in the layers dose not diverge indicating a stable network.

As the theory section on capsule networks explains, the capsules learns poses as well as features. This fact becomes clear from our results. If we look at the triangular signs in figure 4.18 then we see that the reconstructed images are rotated to the same position indicating that the network handles different viewpoints. By examining figure 4.19 we see that there are dimensions of the capsule vectors handling poses such as contrast, color intensity and blurriness. Since the data set consists of image classes that are similar in shape and vary more in color shifts, i.e the background

is different or the image is overall darker or brighter, then we can expect that these poses are going to be represented more in the capsules. Figure 5.1 is from Hinton's paper on dynamic routing [35]. It shows how the corresponding manipulations are done on a capsule network trained on the MNIST [46] data set.

Scale and thickness	
Localized part	
Stroke thickness	
Localized skew	
Width and translation	
Localized part	

**Figure 5.1:** Image from Hinton's paper on dynamic routing. The letters are created by the decoder part of a capsule network trained on the MNIST data set. It displays the different poses that the network encodes. From [35]. Reproduced with permission.

Here we see that the poses like bending, rotation and thickness is represented more. Because the MNIST data set tends to vary more in those features it is understandable that that will be shown in the poses. The conclusion is that the poses of the capsules are being determined based on the varying poses in the data set. This means that the Capsule Network is able to extract and learn different types of poses that are important to differentiate the classes in the data set.

### 5.3 Capsule Network outperforms CNN at the expense of training time

The obtained results clearly show that our implemented Capsule network which gave a test accuracy of 99.62% outperformed our implemented CNN which gave a test accuracy of 98.61%. The main challenge with implementing a good CNN for our problem was finding an optimal network architecture and hyperparameters. For the Capsule network the initial architecture and hyperparameters tuned for the MNIST data set worked well for GTSRB as well, so there the main challenge was instead to counteract overfitting. This shows that the capsule network needed less tuning of network architecture and hyperparameters to perform well on the GTSRB data set.

We also saw that the Capsule Network took approximately 6 hours to train which was significantly longer than the 1.5 hours it approximately took to train the CNN network. One main reason for this is that the Capsule network had approximately 20700000 trainable parameters while the CNN had approximately 10000000 trainable parameters. This shows that one of the main problems with Capsule Networks are the significantly longer training times. In this case we were still able to train the

network in a reasonable amount of time, but for larger data sets this higher time complexity can severely limit the ability to train the network. This demonstrates the need for increasing the efficiency of Capsule Networks, finding ways to optimize the network structure and reduce the number of learnable parameters but still giving good performance. This work has been started by Hinton et. al in [36] where they implement an alternative representation of the pose of an entity as a matrix with  $n$  elements instead of a vector with length  $n$  and therefore reduce the number of parameters in the transformation matrices from  $n^2$  to  $n$ .

## 5.4 Intraclass homogeneity of GTSRB could contribute to the high accuracy

The complexity of classification depends, amongst other things, on intraclass homogeneity and interclass heterogeneity. The former more simply put means that all images of traffic signs with the same class should look rather similar. An example where this would not be the case would be if the data set also had a class called "Dogs", since a dog can lie down, stand, jump, look right/left etcetera and still be considered a dog. Even though they all are dogs, to a computer they would look very different because the pixel values are completely different. The dog class would thus have low intraclass homogeneity. However, a traffic sign is specifically designed to always look the same, suggesting that the intraclass homogeneity of the GTSRB data set is high.

To test this property quantitatively we calculated the five nearest neighbors in L1 and L2 norm to all images in the GTSRB test set. We also saved the classes of those neighbors, information which will be used later in this section. The result of this analysis is illustrated in figure 5.2 and shows that the intraclass homogeneity is very high - in some cases it is even hard to see with the naked eye that the original image and the nearest neighbors are not the same image. This is a consequence of the fact that the GTSRB data set consists of multiple images of the same physical traffic sign but with small differences in rotation, contrast or other minor transformations.

Studying the interclass heterogeneity, which in other words means that images from different classes should look different, is not as straight forward. However it is possible to give a qualitative understanding by looking at the relative intra- versus interclass differences. Or simply: we do not concern ourselves with exactly how big difference it is between different classes, as long as the difference between classes is bigger than the differences within a single class. To some extent the nearest neighbor approach illustrated in figure 5.2 already provides proof of this, since all nearest neighbors are of the same class as the original image. However, we can take this one step further by trying to actually classify an image based on the classes of its  $K$  nearest neighbors. This was done for all images in the GTSRB test. The total accuracy for this approach is presented in table 5.1.



**Figure 5.2:** The 5 nearest neighbors of three randomly chosen images in the GT-SRB test set, calculated in L1-norm. All other observed images gave equivalent results, as well as when calculating nearest neighbors in L2-norm. It is evident that many images in the test set are very alike, suggesting that the intraclass homogeneity is high - which is what to expect since traffic signs of a specific class are specifically created to resemble each other perfectly.

**Table 5.1:** Accuracy for a K-Nearest neighbor classifier on the 12630 images in the GTSRB test set, using L1 or L2 norm.

	Accuracy, L1-norm	Accuracy, L2-norm
<b>1-Nearest neighbor</b>	98.66%	95.95%
<b>5-Nearest neighbor</b>	96.77%	93.06%
<b>10-Nearest neighbor</b>	94.48%	89.62%

As can be seen from table 5.1 a 1-Nearest neighbor classifier, although being very primitive, performs on par with the optimized CNN-model found in this study, and is not far from the state of the art models. Even though this is very interesting, since it tells us that the intraclass differences indeed is smaller than the interclass differences, it is worth mentioning that this is only part of the truth. To be properly compared to those models the K-Nearest neighbor classifier would have to classify images from the test set based on its nearest neighbors in the training set, since the used classifier can not have access to test data during classification. The reason why this more correct approach was not used is due to the second flaw with this classifier - it is very computationally expensive. Every time it is to classify a single image, it needs to look at all images in the training set. Meaning that a four times larger training set takes four times longer to classify with (however, it might mean that the classification becomes more accurate - but we will not go into that here). Neither is the model able to generalize well to images that are too different from the once it has already seen - in contrast to more sophisticated models that tries to

learn characteristic features.

In conclusion we have seen that the GTSRB data set contains rather small differences within each class, at least smaller than relative to other classes. This means that it should be possible to find distinct, well-defined features corresponding to each class that are as good as non-existing in all other classes. In other words it is a data set that would suit a network with convolutional layers perfectly, which we have already seen.

What is more concerning if applied to a real world data is that the K-nearest neighbor classifier gave such good results, since this implies that we are dealing with a very two-dimensional data set. The meaning of this is that the majority of all traffic signs are shown head on. It would thus probably be hard for at least our CNN model to generalize well to traffic signs in different perspectives, such as viewed from below, above or from the side. It is possible that this issue could be mitigated by including perspective augmented images in the training data set, something that most definitely must be considered before implementing our model in any real life application. To put in other words: an autonomous car whose cameras always have to be head on the traffic signs is not ideal! The Capsule network is however designed to handle this multi-perspective scenario so it should in theory be more robust to this than the CNN.

### 5.5 Augmented data lead to great improvement of the Faster R-CNN

A major issue during the detection and classification task of GTSDB was a lack of sufficient computing power. As Faster R-CNN is an advanced network and the input pictures are large the training was very slow. Using our setup training up to 200000 steps took approximately little over a week in total. This prohibited us from training beyond 250000 steps as well as exploring different choices of hyperparameters. The issue of long training times is not unique to our task and will be an issue when applying this or similar designs to real world problems.

As mentioned in the result section after training on the original GTSDB up to 150000 steps the network was able to find most of the signs, but had a hard time correctly predicting which class they belong to. The issue was the same, although to an lower extent, after training for an additional 100000 steps on the extended set. It was also clear that it was rare for the network to predict the presence of a sign in regions without a sign belonging to one of the classes. A strong evidence for this is the PR-curve in figure 4.21b. Both versions of the combined network performed significantly better than their corresponding Faster R-CNN. Since the combined classifier uses a Faster R-CNN to generate bounding boxes, this shows that most of the predicted boxes are of actual signs, otherwise the CNN would not

be able to improve the results.

Another important result is the fact that the performance of the Faster R-CNN trained on the original set reaches its maximum around 100000 steps, while still having issues with classifications as discussed above. This can be interpreted as the training set being too small, with some classes underrepresented to such an extent that the network is unable to learn how to classify these classes correctly. It is always important when working with deep learning techniques to have a data set that is large enough and if that is not the case it can be very challenging to train the network to generalize properly.

When training on the extended set the performance improved further and reached an mAP-score above 0.56 when training was terminated. The fact that extending the set improved the performance to such an extent is another indication that the GTSDb training set is too small to work well for Faster R-CNN. This should not be very surprising as the set was not created for a detection and classification task, but for a pure detection task as mentioned in the method.

Our main conclusion from the results of GTSDb task is that the hardest part was the classification rather than detection. One could speculate that the reason for this is that many of the signs look rather similar. That is the interclass heterogeneity is low, at least with regard to how many images there is in the training set. Had the task involved classes that are more different, for instance finding cars, humans and cats in pictures the classification might have been a smaller issue.

We have also drawn conclusions about how the result could have been improved and we have identified four different approaches. First of all, when training on the extended set we never reached a point when the performance stopped improving. Therefore training should be continued until reaching a point where the mAP-score stops improving. Secondly, even though it is clear that extending the set helped improve the performance it is not clear the method used for extending the set is optimal. After training on the extended set the network still struggles with the same issue as before, only to a lower extent. Therefore it is possible that the performance had benefited from extending the set further and not just doubling it. Without actually testing the impact of a more aggressively extended set it is very hard to give any confident predictions, but given the success of making the set twice as large it seems likely that it would help. It would also be of interest to investigate further how the new images should be generated to yield the best results. In this report we only tested one way of extending the training set and most likely the algorithm could be improved to generate images more true to the originals. We think this is a very promising approach for further work on improving the performance. The third possible method is to investigate how different choices of hyperparameters could have improved our results. As stated in the discussion chapter the time constraint and limited computing power prohibited us from testing more than a single set of hyperparameters so this area is completely uninvestigated by us. The last approach that we would like to propose is to test other models than the specific Faster R-

CNN model that we decided to use. TensorFlow Detection Model Zoo provides many other models, including alternative implementations of Faster R-CNN.

Most of the proposed improvement above requires increased computing power. One solution would be to further investigate the possibilities of Google Cloud computing. As of now, we only used the CPU, since it was the the cheapest and easiest computing instance. We were not able to get Tensorflow to run efficiently with GPU-instructions on Google cloud. If we had, the training time would have decreased substantially.



# 6

## Conclusion

We achieved our aim to implement neural networks capable of detecting and classifying traffic signs. The main focus was the classification task where we implemented two different networks, namely a conventional CNN and a Capsule Network. With these we achieved test accuracies of 98.61% for the CNN and 99.62% for the Capsule Network. Our results would have placed us in the second and fifth place of the results table corresponding to the official challenge on GTSRB, the best result being 99.71%. We were able to beat human performance on the benchmark, which corresponds to an accuracy of 98.84% [7]. Our results show that Capsule Networks are able to achieve state of the art performance on GTSRB, which in several aspects is more complex than the MNIST set which was the main data set used in the original paper by Hinton [35]. This shows that Capsule Networks are able to perform well on data sets with larger number of classes, 43 instead of 10, and with images that have more complex features as well as different resolutions and contrasts. For the Faster R-CNN we were able to train an existing network structure, pretrained on the COCO data set, to detect and classify pictures from the GTSDB data set - showing that transfer learning is a useful tool when taking on new tasks. The model achieved an mAP of 0.56. It provided good detection, but lacked satisfactory classification accuracy. Because of this we combined this network with our CNN to improve the classification part which gave even better results. It would further probably be possible to increase this score if training was to be continued, since it did not show any signs of being saturated.

One thing worth mentioning is that our results highlight the importance of having good data when training machine learning models. The great improvement of the Faster R-CNN after enlarging the GTSDB data set is a prime example of this. With the data society we are living in, large amounts of data is gathered in more and more fields which is a main reason for the rapid increase of machine learning applications. There is still a lot more ground to cover, especially in finding ways to efficiently use the designed models in practice.

The next step of our work could have been to try using a Capsule Network for object detection. This could be a good idea since capsules are designed to keep track of properties like position of the entities in a picture. While Faster R-CNN is frequently used for this purpose, the newer Capsule Network has not been studied

## 6. Conclusion

---

deeper regarding this task. Another suggestion would be to integrate the networks into a real world application, e.g a system which can register the coordinates of all traffic signs along a route from street view.

# Bibliography

- [1] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, pp. 436–444, May 2015. [Online]. Available: <http://dx.doi.org/10.1038/nature14539> Accessed on: May 12, 2018.
- [2] B. H. Frank, “Google brain chief: Deep learning takes at least 100,000 examples,” *VentureBeat*, Oct 23, 2017. [Online]. Available: <https://venturebeat.com/2017/10/23/google-brain-chief-says-100000-examples-is-enough-data-for-deep-learning/> Accessed on: May 12, 2018.
- [3] G. E. Hinton, S. Osindero, and Y.-W. Teh, “A fast learning algorithm for deep belief nets,” *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, Jul. 2006. [Online]. Available: <http://dx.doi.org/10.1162/neco.2006.18.7.1527> Accessed on: May 12, 2018.
- [4] Z. Meng, X. Fan, X. Chen, M. Chen, and Y. Tong, “Detecting small signs from large images,” *CoRR*, vol. abs/1706.08574, Jun. 2017. [Online]. Available: <http://arxiv.org/abs/1706.08574> Accessed on: May 12, 2018.
- [5] D. Cireşan, U. Meier, J. Masci, and J. Schmidhuber, “Multi-column deep neural network for traffic sign classification,” *Neural Networks*, vol. 32, pp. 333–338, Aug. 2012. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0893608012000524> Accessed on: May 12, 2018.
- [6] Álvaro Arcos-García, J. A. Álvarez-García Luis, and M. Soria-Morillo, “Deep neural network for traffic sign recognition systems: An analysis of spatial transformers and stochastic optimisation methods,” *Neural Networks*, vol. Volume 99, pp. 158–165, Mar. 2018. [Online]. Available: <https://doi.org/10.1016/j.neunet.2018.01.005> Accessed on: May 12, 2018.
- [7] J. S. C. I. J. Stallkamp, M. Schlipsing, “Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition,” *Neural Networks*, vol. vol. 32, pp. pp. 323–332, Aug. 2012. [Online]. Available: <https://doi.org/10.1016/j.neunet.2012.02.016> Accessed on: May 12, 2018.
- [8] Ruhr-Universität Bochum - Institut für Neuroinformatik, Bochum, Germany, “INI Benchmark GTSRB Results.” [Online]. Available: <http://benchmark.ini.rub.de/?section=gtsrb&subsection=results> Accessed on: May 13, 2018.

- [9] D. Lu and Q. Weng, “A survey of image classification methods and techniques for improving classification performance,” *International Journal of Remote Sensing*, vol. 28, no. 5, pp. 823–870, Mar. 2007. [Online]. Available: <https://doi.org/10.1080/01431160600746456> Accessed on: May 12, 2018.
- [10] I. Evtimov, K. Eykholt, E. Fernandes, T. Kohno, B. Li, A. Prakash, A. Rahmati, and D. Song, “Robust physical-world attacks on machine learning models,” *CoRR*, vol. abs/1707.08945, Apr. 2017. [Online]. Available: <http://arxiv.org/abs/1707.08945> Accessed on: May 12, 2018.
- [11] C. Sitawarin, A. N. Bhagoji, A. Mosenia, P. Mittal, and M. Chiang, “Rogue signs: Deceiving traffic sign recognition with malicious ads and logos,” *CoRR*, vol. abs/1801.02780, Mar. 2018. [Online]. Available: <http://arxiv.org/abs/1801.02780> Accessed on: May 12, 2018.
- [12] I. Goodfellow, Y. Bengio, and A. Courville, “Introduction,” in *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016, ch. 1, pp. 1–26. [Online]. Available: <http://www.deeplearningbook.org> Accessed on: May 12, 2018.
- [13] V. N. Vapnik, “Neural networks,” in *Statistical Learning Theory*. New York, NY, USA: Wiley-Interscience, 1998, ch. 9.6, p. 395.
- [14] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015. [Online]. Available: <http://neuralnetworksanddeeplearning.com/> Accessed on: May 12, 2018.
- [15] Stanford University, Stanford, CA, USA, “Cs231n: Convolutional neural networks for visual recognition.” [Online]. Available: <http://cs231n.github.io/neural-networks-1/> Accessed on: May 12, 2018.
- [16] P. Ramachandran, B. Zoph, and Q. V. Le, “Searching for activation functions,” *CoRR*, vol. abs/1710.05941, Oct. 2017. [Online]. Available: <http://arxiv.org/abs/1710.05941> Accessed on: May 12, 2018.
- [17] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, pp. 533–536, Oct. 1986. [Online]. Available: <http://dx.doi.org/10.1038/323533a0> Accessed on: May 12, 2018.
- [18] Stanford University, Stanford, CA, USA, “Cs231n: Convolutional neural networks for visual recognition.” [Online]. Available: <http://cs231n.github.io/optimization-1/> Accessed on: May 12, 2018.
- [19] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng, “On optimization methods for deep learning,” in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, 2011, pp. 265–272. [Online]. Available: <http://dl.acm.org/citation.cfm?id=3104482.3104516> Accessed on: May 12, 2018.
- [20] S. Ruder, “An overview of gradient descent optimization algorithms,” *CoRR*, vol. abs/1609.04747, Jun. 2017. [Online]. Available: <http://arxiv.org/abs/1609.04747> Accessed on: May 12, 2018.

- 
- [21] A. Géron, “The machine learning landscape,” in *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 1st ed. Sebastopol, CA, USA: O’Reilly Media, Inc., 2017, ch. 1, pp. 3–32.
- [22] J. Kukacka, V. Golkov, and D. Cremers, “Regularization for deep learning: A taxonomy,” *CoRR*, vol. abs/1710.10686, Oct. 2017. [Online]. Available: <http://arxiv.org/abs/1710.10686> Accessed on: May 13, 2018.
- [23] A. Géron, “Training deep neural nets,” in *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 1st ed. Sebastopol, CA, USA: O’Reilly Media, Inc., 2017, ch. 11, pp. 277–316.
- [24] Chabacano, “Overfitting,” 2008, [Electronic image]. [Online]. Available: <https://commons.wikimedia.org/w/index.php?curid=3610704> Accessed on: May 13, 2018.
- [25] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, Jun. 2014. [Online]. Available: <http://jmlr.org/papers/v15/srivastava14a.html> Accessed on: May 13, 2018.
- [26] H. Shimodaira, “Improving predictive inference under covariate shift by weighting the log-likelihood function,” *Neural Networks*, vol. 90, pp. 227–244, Oct. 2000. [Online]. Available: [https://doi.org/10.1016/S0378-3758\(00\)00115-4](https://doi.org/10.1016/S0378-3758(00)00115-4) Accessed on: May 13, 2018.
- [27] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *CoRR*, vol. abs/1502.03167, Mar. 2015. [Online]. Available: <http://arxiv.org/abs/1502.03167> Accessed on: May 13, 2018.
- [28] S. C. Wong, A. Gatt, V. Stamatescu, and M. D. McDonnell, “Understanding data augmentation for classification: when to warp?” *CoRR*, vol. abs/1609.08764, Nov. 2016. [Online]. Available: <http://arxiv.org/abs/1609.08764> Accessed on: May 13, 2018.
- [29] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.
- [30] Stanford University, Stanford, CA, USA, “Cs231n: Convolutional neural networks for visual recognition.” [Online]. Available: <http://cs231n.github.io/convolutional-networks/> Accessed on: May 12, 2018.
- [31] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, Apr 2011, pp. 315–323. [Online]. Available: <http://proceedings.mlr.press/v15/glorot11a.html> Accessed on: May 13, 2018.

- [32] I. Goodfellow, Y. Bengio, and A. Courville, “Convolutional networks,” in *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016, ch. 9, pp. 326–366. [Online]. Available: <http://www.deeplearningbook.org> Accessed on: May 12, 2018.
- [33] C. M. Bishop, “Neural networks,” in *Pattern Recognition and Machine Learning*. Berlin, Germany: Springer Science+Business Media, 2006, ch. Neural Networks, pp. 225–290.
- [34] Stanford University, Stanford, CA, USA, “Cs231n: Convolutional neural networks for visual recognition.” [Online]. Available: <http://cs231n.github.io/linear-classify/> Accessed on: May 12, 2018.
- [35] S. Sabour, N. Frosst, and G. E. Hinton, “Dynamic routing between capsules,” *CoRR*, vol. abs/1710.09829, Oct. 2017. [Online]. Available: <http://arxiv.org/abs/1710.09829> Accessed on: May 13, 2018.
- [36] G. E. Hinton, S. Sabour, and N. Frosst, “Matrix capsules with EM routing,” in *International Conference on Learning Representations*, Mar. 2018. [Online]. Available: <https://openreview.net/forum?id=HJWLfGWRb> Accessed on: May 13, 2018.
- [37] G. E. Hinton, A. Krizhevsky, and S. D. Wang, “Transforming auto-encoders,” in *Artificial Neural Networks and Machine Learning – ICANN 2011*, Espoo, Finland, 2011, pp. 44–51.
- [38] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards real-time object detection with region proposal networks,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017.
- [39] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders, “Selective search for object recognition,” *International Journal of Computer Vision*, vol. 104, no. 2, pp. 154–171, Sep. 2013. [Online]. Available: <https://ivi.fnwi.uva.nl/isis/publications/2013/UijlingsIJCV2013> Accessed on: May 12, 2018.
- [40] R. B. Girshick, “Fast R-CNN,” *CoRR*, vol. abs/1504.08083, Sep. 2015. [Online]. Available: <http://arxiv.org/abs/1504.08083> Accessed on: May 13, 2018.
- [41] A. Géron, “Classification,” in *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 1st ed. Sebastopol, CA, USA: O’Reilly Media, Inc., 2017, ch. 3, pp. 81–106.
- [42] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes (VOC) challenge,” *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, Jun. 2010. [Online]. Available: <https://doi.org/10.1007/s11263-009-0275-4> Accessed on: May 12, 2018.
- [43] “Glossary of terms,” *Machine Learning*, vol. 30, no. 2, pp. 271–274, Feb. 1998. [Online]. Available: <https://doi.org/10.1023/A:1017181826899> Accessed on: May 13, 2018.

- [44] K. Pearson F.R.S., “On lines and planes of closest fit to systems of points in space,” *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901. [Online]. Available: <https://doi.org/10.1080/14786440109462720> Accessed on: May 13, 2018.
- [45] L. van der Maaten and G. Hinton, “Visualizing data using t-SNE,” *Journal of Machine Learning Research*, vol. abs/2579-2605, Aug. 2008. [Online]. Available: <http://jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf> Accessed on: May 13, 2018.
- [46] Y. LeCun, C. Cortes, and C. J. Burges, “The MNIST database of handwritten digits.” [Online]. Available: <http://yann.lecun.com/exdb/mnist/> Accessed on: May 12, 2018.
- [47] A. Géron, “Training models,” in *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 1st ed. Sebastopol, CA, USA: O’Reilly Media, Inc., 2017, ch. 4, pp. 107–146.
- [48] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, “Self-normalizing neural networks,” *CoRR*, vol. abs/1706.02515, Sep. 2017. [Online]. Available: <http://arxiv.org/abs/1706.02515> Accessed on: May 13 2018.
- [49] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft COCO: common objects in context,” *CoRR*, vol. abs/1405.0312, Feb. 2015. [Online]. Available: <http://arxiv.org/abs/1405.0312> Accessed on: May 14, 2018.
- [50] L. Y. Pratt, J. Mostow, and C. A. Kam, “Direct transfer of learned information among neural networks,” in *The Ninth National Conference on Artificial Intelligence*, 1991, pp. 584–589. [Online]. Available: <https://www.aaai.org/Papers/AAAI/1991/AAAI91-091.pdf> Accessed on: May 13 2018.
- [51] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, “The German Traffic Sign Recognition Benchmark: A multi-class classification competition,” in *IEEE International Joint Conference on Neural Networks*, San José, CA, USA, 2011, pp. 1453–1460.
- [52] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel, “Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark,” in *International Joint Conference on Neural Networks*, Dallas, TX, USA, 2013, pp. 1–8.

# A

## Configuration file for Faster R-CNN

```
# Faster R-CNN with Inception v2, configuration for MSCOCO Dataset.
# Users should configure the fine_tune_checkpoint field in the train config as
# well as the label_map_path and input_path fields in the train_input_reader and
# eval_input_reader. Search for "PATH_TO_BE_CONFIGURED" to find the fields that
# should be configured.
```

```
model {
  faster_rcnn {
    num_classes: 43 #changed to number of classes in GTSDb task
    image_resizer {
      keep_aspect_ratio_resizer {
        min_dimension: 600
        max_dimension: 1024
      }
    }
    feature_extractor {
      type: 'faster_rcnn_inception_v2'
      first_stage_features_stride: 16
    }
    first_stage_anchor_generator {
      grid_anchor_generator {
        scales: [0.25, 0.5, 1.0, 2.0]
        aspect_ratios: [1.0]
        height_stride: 16
        width_stride: 16
      }
    }
    first_stage_box_predictor_conv_hyperparams {
      op: CONV
      regularizer {
        l2_regularizer {
          weight: 0.0
        }
      }
    }
    initializer {
      truncated_normal_initializer {
        stddev: 0.01
      }
    }
  }
}
```



## A. Configuration file for Faster R-CNN

---

```
    }
  }
}
first_stage_nms_score_threshold: 0.0
first_stage_nms_iou_threshold: 0.7
first_stage_max_proposals: 300
first_stage_localization_loss_weight: 2.0
first_stage_objectness_loss_weight: 1.0
initial_crop_size: 14
maxpool_kernel_size: 2
maxpool_stride: 2
second_stage_box_predictor {
  mask_rcnn_box_predictor {
    use_dropout: false
    dropout_keep_probability: 1.0
    fc_hyperparams {
      op: FC
      regularizer {
        l2_regularizer {
          weight: 0.0
        }
      }
      initializer {
        variance_scaling_initializer {
          factor: 1.0
          uniform: true
          mode: FAN_AVG
        }
      }
    }
  }
}
}
}
second_stage_post_processing {
  batch_non_max_suppression {
    score_threshold: 0.0
    iou_threshold: 0.6
    max_detections_per_class: 100
    max_total_detections: 300
  }
  score_converter: SOFTMAX
}
second_stage_localization_loss_weight: 2.0
second_stage_classification_loss_weight: 1.0
}

train_config: {
  batch_size: 1
  optimizer {
    momentum_optimizer: {
      learning_rate: {
        manual_step_learning_rate {
          initial_learning_rate: 0.0002
          schedule {
            step: 0
            learning_rate: .0002
          }
        }
      }
    }
  }
}
```

```
    }
    schedule {
      step: 900000
      learning_rate: .0002
    }
    schedule {
      step: 1200000
      learning_rate: .0002
    }
  }
}
momentum_optimizer_value: 0.9
}
use_moving_average: false
}
gradient_clipping_by_norm: 10.0
fine_tune_checkpoint: "gs://bachelor-belgium/models/model.ckpt"
from_detection_checkpoint: true
# Note: The below line limits the training process to 200K steps, which we
# empirically found to be sufficient enough to train the COCO dataset. This
# effectively bypasses the learning rate schedule (the learning rate will
# never decay). Remove the below line to train indefinitely.
num_steps: 250000
data_augmentation_options {
  random_horizontal_flip {
  }
}
}
}

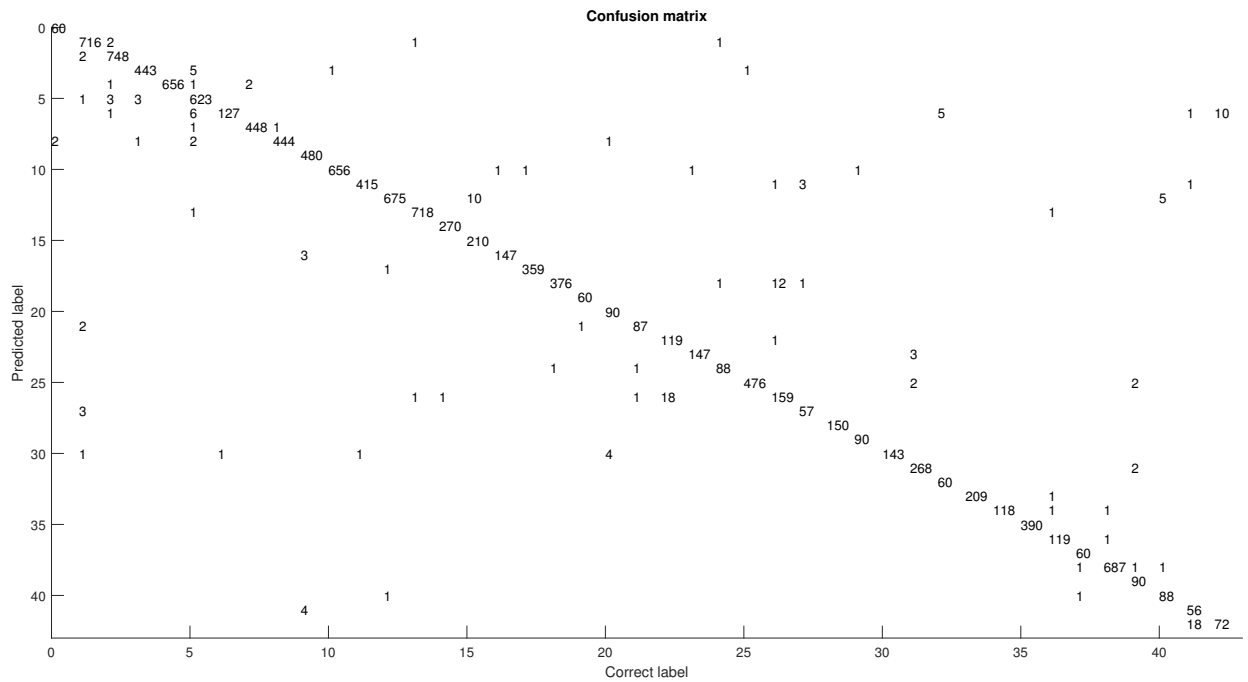
train_input_reader: {
  tf_record_input_reader {
    input_path: "gs://bachelor-belgium/data/TrainGTSDB_ext.record"
  }
  label_map_path: "gs://bachelor-belgium/data/GTSDB_label_map.pbtxt"
}

eval_config: {
  num_examples: 235
  # Note: The below line limits the evaluation process to 10 evaluations.
  # Remove the below line to evaluate indefinitely.
  max_evals: 10
}

eval_input_reader: {
  tf_record_input_reader {
    input_path: "gs://bachelor-belgium/data/TestGTSDB.record"
  }
  label_map_path: "gs://bachelor-belgium/data/GTSDB_label_map.pbtxt"
  shuffle: false
  num_readers: 1
}
```

# B

## **Additional CNN results**



**Figure B.1:** Confusion matrix of the final model. Each row represents the predicted class and the columns indicate the true label. The network was trained on the extended data set with batch size 15, a learn rate of 0.005 and dropout rate of 0.37% for 75 epochs.



**Figure B.2:** 25 randomly selected images from the set of misclassified images of the final CNN, each with its predicted class and the accuracy score on how certain the network is on that prediction.