CHALMERS
UNIVERSITY OF TECHNOLOGY

Deep Learning Methods and Applications

Classification of Traffic Signs and Detection of Alzheimer's Disease from Images

Master's thesis in Communication Engineering and Biomedical Engineering

LINNÉA CLAESSON
BJÖRN HANSSON

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2017
EX004/2017

Master's thesis EX004/2017

Deep Learning Methods and Applications

Classification of Traffic Signs and
Detection of Alzheimer's Disease from Images

LINNÉA CLAESSON
BJÖRN HANSSON

Supervisor and Examiner: Prof. Irene Y.H. Gu

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Signals and Systems
Division of Signal Processing and Biomedical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2017

Deep Learning Methods and Applications:
Classification of Traffic Signs and
Detection of Alzheimer's Disease
LINNÉA CLAESSON, BJÖRN HANSSON
for the Alzheimer's Disease Neuroimaging initiative*

Supervisor and Examiner: Prof. Irene Y.H. Gu, Signals and Systems

Master's Thesis EX004/2017
Department of Signals and Systems
Division of Signal Processing and Biomedical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

*Data used in preparation of this article were obtained from the Alzheimer's Disease Neuroimaging Initiative (ADNI) database (adni.loni.usc.edu). As such, the investigators within the ADNI contributed to the design and implementation of ADNI and/or provided data but did not participate in analysis or writing of this report. A complete listing of ADNI investigators can be found at: `http://adni.loni.usc.edu/wp-content/uploads/how_to_apply/ADNI_Acknowledgement_List.pdf`

Typeset in LaTeX
Gothenburg, Sweden 2017

Deep Learning Methods and Applications
Classification of Traffic Signs and Detection of Alzheimer's Disease from Images
LINNÉA CLAESSON, BJÖRN HANSSON
Department of Signals and Systems
Chalmers University of Technology

Abstract

In this thesis, the deep learning method convolutional neural networks (CNNs) has been used in an attempt to solve two classification problems, namely traffic sign recognition and Alzheimer's disease detection. The two datasets used are from the German Traffic Sign Recognition Benchmark (GTSRB) and the Alzheimer's Disease Neuroimaging Initiative (ADNI). The final test results on the traffic sign dataset generated a classification accuracy of 98.81 %, almost as high as human performance on the same dataset, 98.84 %. Different parameter settings of the selected CNN structure have also been tested in order to see their impact on the classification accuracy. Trying to distinguish between MRI images of healthy brains and brains afflicted with Alzheimer's disease gained only about 65 % classification accuracy. These results show that the convolutional neural network approach is very promising for classifying traffic signs, but more work needs to be done when working with the more complex problem of detecting Alzheimer's disease.

Keywords: Convolutional neural networks, deep learning, machine learning, traffic sign recognition, Alzheimer's disease detection, GTSRB, ADNI, CNN

Acknowledgements

We would firstly like to express our sincerest gratitude to our supervisor Irene Yu-Hua Gu at the department of Signals and Systems at Chalmers University, where this thesis has been conducted. We would like to thank her for her help and guidance throughout this work.

We are also immensely thankful for our partners, friends, and family who have always supported and encouraged us, not just throughout this work, but through all of our time at university. We never would have made it this far without you.

Additionally, we would also like to express our thanks to the German Traffic Sign Recognition Benchmark and the Alzheimer's Disease Neuroimaging Initiative for making their datasets publicly available to stimulate research and development.

We have matured both academically and personally from this experience, and are very grateful for having had the opportunity to help further research in this exciting field.

Linnéa Claesson, Björn Hansson, Gothenburg, January 2017

Contents

List of Figures

List of Tables

1
Introduction

Machine learning has in recent years gained an upswing in the amount of interest it has received, specifically the subfield that is deep learning. The industry is screaming for knowledge and expertise, and universities are not far behind. More and more start offering various machine learning or artificial intelligence courses, to meet the demands of the industry and interest of students.

This work aims to contribute further to the field of deep learning, by exploring the possibilities of using it to classify image data.

1.1 Background

This thesis has been conducted at Chalmers University of Technology, at the department of Signals and Systems, in Gothenburg. It investigates deep learning methods and their applications. Main focus has been on classifying traffic signs, using convolutional neural networks (CNNs) and analysis of the performance. Such a system can be used for both autonomous and assisted driving.

To further investigate the performance and capability of CNNs, another dataset consisting of Magnetic Resonance Images of both healthy brains and brains with Alzheimer's disease was used. This was done in order to investigate whether CNNs can be trained to detect if a person has Alzheimer's disease or not.

1.2 Goals

This thesis aims to explore the field of deep learning, specifically CNNs. The goal was to build two well-performing systems, which both make use of CNNs. One to classify traffic signs, and one to distinguish between healthy brain images, and images of brains with Alzheimer's disease.

1.3 Constraints

CNNs can be used for other purposes than classification, e.g. segmentation, which can be seen as another form of classification. For the sake of this study however, only standard classification has been taken into consideration, due to time restrictions.
For the traffic sign recognition system, one constraint that has been set is to use images which have been constructed in such a way that they contain one traffic sign

only. The sign is centered and takes up most of the space of the image, i.e. the problem of *detecting* traffic signs has already been solved earlier when the dataset was created.

The aim of studying the performance on the Alzheimer's dataset was not to create a perfect solution, but to examine whether it might be possible to detect Alzheimer's disease using CNNs.

1.4 Problem Formulation

This report aims to investigate the field of deep learning by answering the following questions:

- How well can CNNs perform on traffic sign recognition?
- Is it possible to use CNNs to detect Alzheimer's disease from brain images?
- What are the main advantages and disadvantages of CNNs?
- What is the impact of changing the hyperparameters of a CNN?

1.5 Disposition

The disposition of this report generally follows that of a standard technical report. Section 2 relays the background and theory necessary to understand the framework of the performed study. The experimental setup can then be found in section 3. Since the work has been carried out mainly in two parts, one concerning traffic sign recognition and one on detection of Alzheimer's disease, they have been granted one section each, namely sections 4 and 5 respectively. Both sections follows the same structure, starting with an explanation of the methods used, presentation of experimental results and performance evaluation, and finally a general discussion of the results. Thereafter follows a section discussing the important questions of ethics and sustainability the technology raises, section 6. Finally in section 7, conclusions drawn from this study are presented.

2
Background

This section aims to describe both the theory needed to fully understand the work conducted, as well as introduce related work that can be found interesting for this thesis. It starts off with introducing machine learning and the theory this work is built upon. Thereafter, a section on autonomous driving and the challenges of traffic sign recognition today follows, along with a short description of detection of Alzheimer's disease from MRI images. Lastly, different software libraries useful for implementing machine learning algorithms, as well as what to consider when choosing the appropriate hardware for these kinds of problems are discussed.

2.1 Machine Learning and Deep Learning

Machine learning has been at the foundation of this thesis, particularly deep learning and CNNs. The theory and necessary background information needed to understand the work is described in this section, starting with a general introduction to machine learning and then building upon that to eventually explain how CNNs operate.

2.1.1 General Introduction

Machine learning is a subfield of artificial intelligence which is becoming increasingly more popular, and is widely used out in the industry to solve various tasks. However, artificial intelligence is not a new term within computer science, it all started when Alan Turing proposed the question "Can machines think?"[9]. Since Turing came up with his Imitation Game, the focus of artificial intelligence has shifted around between various areas. Given the enormous amounts of data available today, it is no wonder that the data-driven approach of machine learning has become so popular.

So, what constitutes learning for a machine? Mitchell in his book defines learning as follows: *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E"*[10]. To put this into context, the classical spam filter example can be used. The task here is to predict if an email is spam, the experience is the data set used for training, and performance can be measured as the ratio of correctly classified emails. Other popular areas of use are recommender systems, "If you liked this, you might also like this...", and social networking sites also use machine learning techniques to, for example, predict people you might know on the site.

2.1.2 Neural Networks

One area of machine learning that has oscillated in popularity over the years since the 1940's, and gained a recent upswing, is neural networks[11]. They are inspired by the biological neural networks in the brain and try to mimic their behaviour. Neural networks consist of an input, one or more hidden layers, and one output layer. When one talks about an N-layer neural network, what is generally referred to is the number of hidden layers plus the output layer. In a feed-forward network, the input does not perform any computations and does therefore not count as a layer[11]. An example of a three-layer neural network can be seen in figure 2.1.

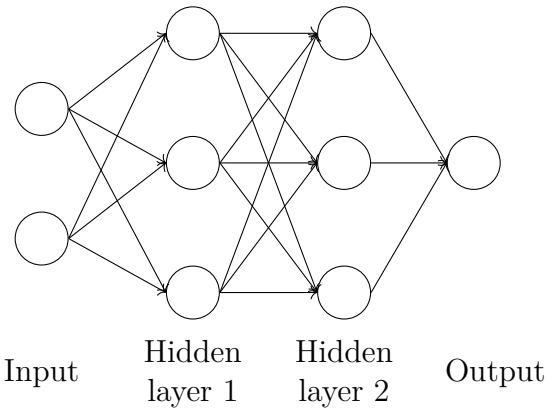


Figure 2.1: Example of a neural network with two hidden layers.

The neurons in a neural networks are fully connected and all have learnable weights and biases. Neural networks are capable of approximating non-linear functions, but are basically just a black box between the input and output and are therefore difficult to analyse. They also do not scale well for the use of images as input, since the number of weights would increase drastically because each pixel would count as a neuron in the input layer.

2.1.3 CNNs

One type of neural network that specialises in input data with a grid-like structure, such as images, are CNNs. They have been proven tremendously successful in practical applications. As the name indicates, the mathematical operation called convolution is used in at least one of its layers, instead of general matrix multiplication[12].

2.1.3.1 Workings of a CNN

CNNs are very similar to regular neural networks, but arranges its neurons in three dimensions – width, height, and depth. A neuron inside a layer is also only connected to a small region of the layer before it, called the receptive field, and not fully connected as in a regular neural network.

The architecture of CNNs consists of several different types of sequential layers, some of which will also be repeated. Below are some of the most common types described:

Convolutional layer As the name implies, this is the core building block of a CNN. It consists of a set of filters that are convolved across the width and height dimensions of the image. The filters with which the image is convolved has the same number of dimensions as the image, each with the same depth (e.g. three if RGB image) but smaller width and height, commonly used spatial sizes are e.g. 3×3 or 5×5. The output width and height depends on the size of the filter, the stride (number of pixels the filter is moved between each computation, usually one or two), and the amount of zero-padding around the image. The output depth will be the same as the number of filters applied.

The convolution process supports three ideas that can help improve a machine learning system, namely sparse interactions, parameter sharing, and equivariant representation[12]. Additionally, it also to some degree makes it invariant to shifts, scaling, and distortions[3].

The output from a convolution of the input and one filter is called a feature map, or sometimes an activation map. There will be one feature map generated by each filter in the layer, and together they make up the output depth. The spatial size of each feature map is dependent on the input image size, padding, filter size, and stride. The fact that the filter is smaller than the input leads to sparse interactions. Each unit on a feature map has n^2 connections to an $n \times n$ area in the input, called the receptive area. Compare this with regular neural networks, where every input is connected to every output. For example with image processing, this means that small, meaningful features, such as edges, can be detected and fewer parameters need to be stored[12].

Each unit on the feature map has n^2 trainable weights plus a trainable bias. All units on a feature map share these same parameters, it can be interpreted as if a feature map is, as the the name suggests, detecting different features such as horizontal or vertical edges, it makes it independent of where in the input the edges are detected. Instead, it is their relative positioning that is of interest. This parameter sharing saves a significant amount of memory[3]. However, the separate feature maps will not share parameters, since they are detecting different features.

Additionally, this form of parameter sharing in the case of convolution makes the function equivariant to translation, i.e. if the input changes the output will change in the same way[12].

Rectified linear units layer, ReLU Increases non-linearity by applying the element wise non-saturating activation function $f(x) = max(0, x)$. An illustration of how this works can be seen in figure 2.2.

It has been shown that the network can train several times faster using this non-saturating function, as compared to using saturating functions such as $f(x) = tanh(x)$, or the sigmoid function, $f(x) = \frac{1}{1+e^{-x}}$[4]. Spatial size is left unchanged. A small, non-zero gradient, α, for negative numbers can also be

used, as in (2.1).

$$f = \begin{cases} x, & x > 0 \\ \alpha x, & otherwise \end{cases} \tag{2.1}$$

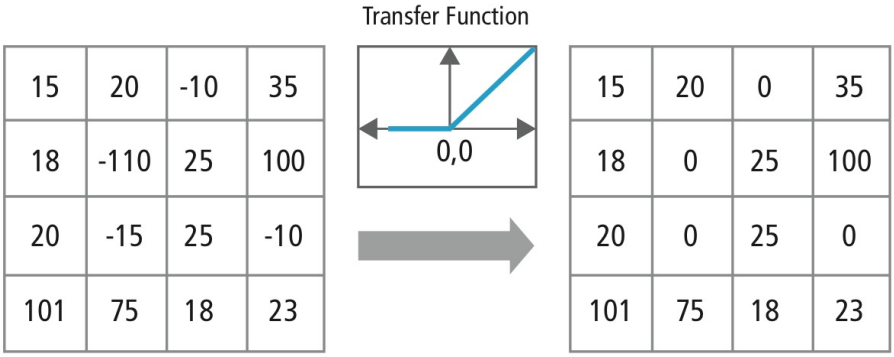


Figure 2.2: Example of how a rectified linear units layer works. All the negative valued numbers in the left box have been set to zero after the rectifier function has been applied, all other values are kept unchanged[1].

Pooling layer Non-linear down sampling of the volume by using small filters to sample for example the maximum or average values in a rectangular area of the output from the previous layer. Pooling reduces the spatial size, to reduce the amount of parameters and computations, and additionally avoids overfitting, i.e. high training accuracy but low validation accuracy. It is displayed in figure 2.3 how pooling layers operate.

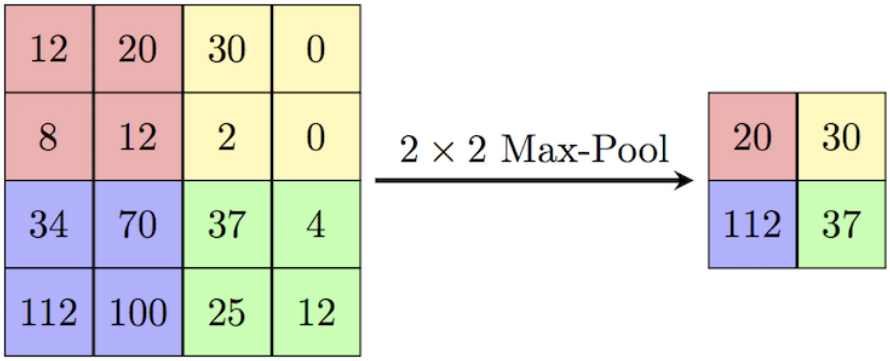


Figure 2.3: Examples of how max pooling operates, the box to the left has been downsampled by taking the maximum value of each 2×2 sub-region[2].

Normalisation layer Different kinds of normalisation layers have been proposed to normalise the data, but have not proven useful in practice and have therefore not gained any solid ground[13].

Fully connected layer Neurons in this layer are fully connected to all activations in the previous layers, as in regular neural networks. These are usually at the end of the network, e.g. outputting the class probabilities.

Loss layer Often the last layer in the network that computes the objective of the task, such as classification, by e.g. applying the softmax function, see equation (2.2).

$$\sigma(z) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad for j = 1, ..., K \tag{2.2}$$

A combination of the described layers can be used to form a CNN architecture. Below can be seen a typical architecture pattern for a CNN[13]:

$$Input \rightarrow [[Conv \rightarrow ReLU] * N \rightarrow Pool?] * M \rightarrow [FC \rightarrow ReLU] * K \rightarrow FC$$

The $*$ represents repetition, N, M, and K are integers greater than zero. N is generally less than or equal to three and K strictly less than three. $Pool?$ indicates that the pooling layer is optional. It is often a good idea to stack more than one convolutional layer before the pooling layer for larger and deeper networks, since the convolutional layer can detect more complex features of the input volume before the destructive pooling operation[13].

It is common to apply dropout during training on the fully connected layers. Dropout rate is a simple way to reduce overfitting. During training, individual nodes are deactivated with a certain probability $1 - p$, or kept active with probability p. The incoming and outgoing connections to a deactivated node are also dropped. In addition to reducing overfitting this also lowers the amount of computations required and allows for better performance. During testing however, all nodes are activated[14].

When initialising the weights of the network, it is important not to set all of them to zero, since this can lead to unwanted symmetry in the updates. Instead it is usually a good idea to set them to small, random numbers, for example by sampling them from a Gaussian distribution.

For training, a loss expression to minimise needs to exist, e.g. by computing categorical cross-entropy between the predictions and targets, as described by equation 2.3. For each instance i, the cross-entropy between the prediction probabilities in p_i, which could be e.g. the softmax output, and target value t_i is calculated. The objective is then to minimise this loss expression during training of the network.

$$L_i = -\sum_j t_{i,j} \log(p_{i,j}) \tag{2.3}$$

2.1.3.2 Existing Networks

Several designs of CNN architectures have already been created, some of them will be described here.

LeNet, 1998 LeCun was first with successfully implementing an application of a CNN, the most notable one being LeNet from 1998 used for handwriting recognition. Figure 2.4 shows the architecture of LeNet-5. It consists of seven layers, not counting the input layer. The input images used were of size 32×32. The first layer consists of six 5×5 filters, which after the convolution brings down the size to 28×28. Following the convolution comes a sub-sampling layer implementing max pooling and then another sixteen 5×5 filters for the second convolution layer, followed by the final sub-sampling layer. The feature maps have now been brought down to a size of 5×5, before entering the fully connected layer[3].

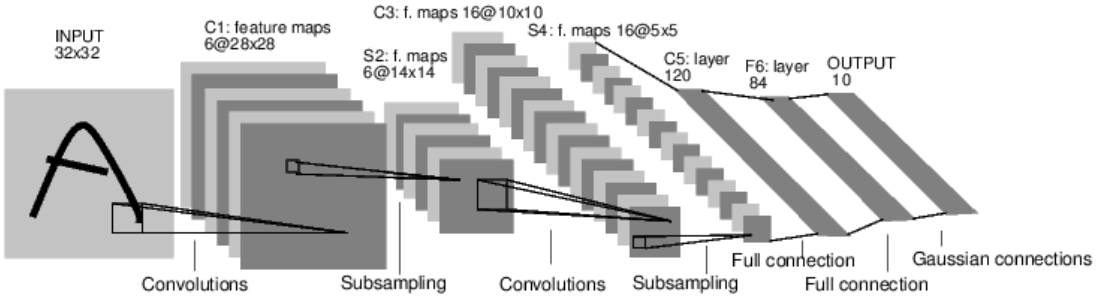


Figure 2.4: Architecture of LeNet-5, 1998[3].

AlexNet, 2012 AlexNet was the winner of the ImageNet ILSVRC challenge in 2012, by a large margin[15]. The architecture for AlexNet can be seen in figure 2.5 (the unfortunate cropping at the top stems from the original article), it was named after Alex Krizhevsky, one of its creators. The input used was images of size 224×224 and the first convolutional layer used 96 filters of size 11×11 with stride four, whereas the rest of the convolutional layers use filters of size 3×3. The full architecture will not be described here, but compared to LeNet the main differences are that it is a bigger and deeper network, it uses ReLU layers, and trained on two GPUs using more data[4]. Noteworthy is also that they used a normalisation layer, which was very popular at the time but is not commonly used anymore[13].

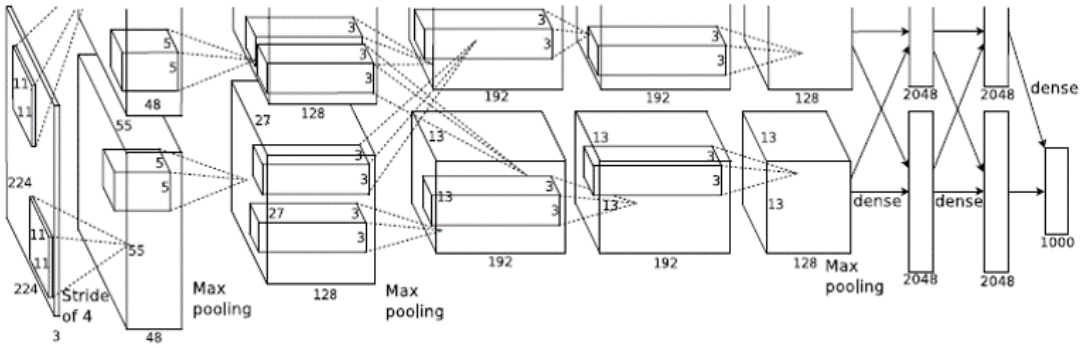


Figure 2.5: Architecture of AlexNet, 2012. The cropping on the top of the image stems from the original article[4].

GoogLeNet, 2014 The winner of the ILSVRC challenge in 2014 was GoogLeNet, a 22 layers deep network. The structure of the network can be seen in figure 2.6. It introduced an inception model, "network in network", and uses a twelfth of the number of parameters AlexNet used[5].

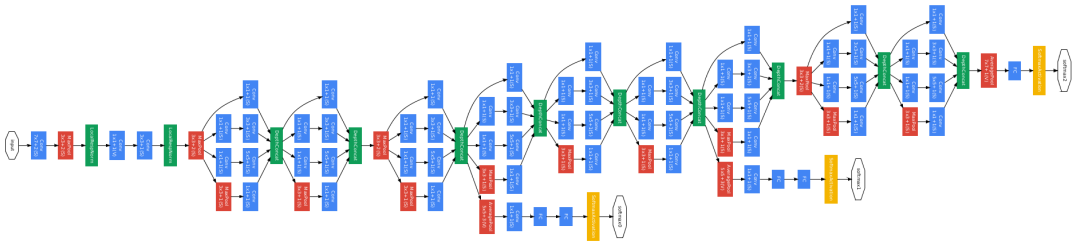


Figure 2.6: Network structure of the very complex GoogleNet[5].

VGGNet, 2014 In figure 2.7 the configuration of VGGNet is shown, which also entered the ILSVRC challenge in 2014 and generalises very well to other data sets[6]. The configurations range from 11 to 19 weight layers, i.e. convolutional and fully connected layers, with a total number of 133 million weights for the smallest configurations, to 144 million weights in the largest. Even though GooGleNet outperformed VGGNet, this is still a very common architecture to use due to it being much less complex than GoogLeNet.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 2.7: Configurations of the CNNs of VGGNet, shown in the columns. The depth increases from left to right and the added layers are shown in bold[6].

2.1.4 3D CNNs

CNNs are usually constructed to be used on two dimensional images, but can be extended to work on three dimensional images, such as on MRI images. It works in basically the same way as with regular two dimensional images, but the filters for the convolutional and pooling layers are extended into three dimensional filters instead of just two, and then the calculations can be carried out in the same manner as before, but now in three dimensions.

2.1.5 Ensemble Learning

To improve upon the results when solving a machine learning problem, an ensemble may be used. Ensembles make use of several individually trained classifiers or models and then combine their results to classify new instances. Basically any type of classification model can be used to create ensembles, such as neural networks or classification trees[16].

Bagging and Boosting are two ensemble techniques. Bagging, or Bootstrap Aggregating, creates an ensemble by training each model individually, with a randomly drawn subset of the training data. For classification, the models then vote with equal weight to determine the classes of the instances. Boosting on the other hand builds entirely new models to try to more accurately classify previously misclassified instances.

2.1.6 Data augmentation

One of the major problems faced when training neural networks is the need for a large amount of training data. Collecting a large amount of data is often very time-consuming and cumbersome. Sometimes it may not even be possible to collect more data and one has to make do with what is available. To solve this problem a method called data augmentation is often used.

Data augmentation means that the same images are used several times but are deformed in various ways to make them different from the original. This means that a dataset can be inflated to several times its original size, which can lead to better training of the network. It has also been shown to reduce overfitting in networks[17].

2.2 Traffic Sign Recognition for Autonomous Vehicles and Assistance Driving Systems

Traffic sign recognition systems are implemented today both to assist human drivers and enable the future of autonomous driving. This section discusses challenges faced by traffic sign recognition and the state of autonomous driving today.

2.2.1 Challenges of Traffic Sign Recognition for Computers

Traffic signs are an essential tool for anyone traveling in modern traffic. They provide direct information on which rules currently apply, and by extension how one is supposed to act and can expect others to act. Traffic signs are very reliable and, as opposed to for example traffic lights, they are not dependent on any external support to operate. This means traffic signs represent a well known and reliable system to provide drivers of vehicles with information, and will remain the standard for the foreseeable future.

Since this is the case, those who strive to create autonomous vehicles capable of safely and efficiently traversing in modern traffic must be able to use this system in order to operate properly. This is a task humans are very well suited for, since traffic signs are designed to be easily recognizable by a human observer[18]. For computers this is a more complex task as there are almost endless variations to how signs will be perceived, depending on angles, light and weather conditions, partial occlusion by other objects etc. This means that a system must be able to handle a large degree of variation in the images supplied, but still provide a correct interpretation, which is no simple task[18].

Computers do however have advantages their human counterparts lack, one of them being the ability to quickly access large databases that contain detailed information about the surroundings they are driving in[7]. With this information they do not only know what the current conditions should be, but also what they can expect further down the road, in theory allowing for better planning. In figure 2.8 an intersection that has been pre-mapped for Google Incs self-driving cars is illustrated. Allowing this information to be shared between vehicles means that there is a continuous update of changes that otherwise would be unknown until directly observed[7].

Regardless of how much these systems know beforehand, all information is at some point recorded for the first time. As such, the systems must be able to handle changes without prior warning. Thus some method of computer vision need to be used to allow for a correct interpretation of the surroundings. There is no universal method for identifying certain objects in images and options include nearest neighbour[19], random forests[20], support vector machines[21][22], and neural networks[23]. All of these are different examples of machine learning techniques used to enable a computer to identify an object.

A system that correctly identifies traffic signs is not just valuable for self-driving cars but can be of assistance to human drivers as well. The ability to quickly and easily find out the speed limit or other rules in place by simply glancing at the

dashboard can make it easier for human drivers, e.g. in case they missed a sign or are simply feeling unsure of what the current situation is. Systems like these are in fact already offered by car manufacturers today[24][25][26].

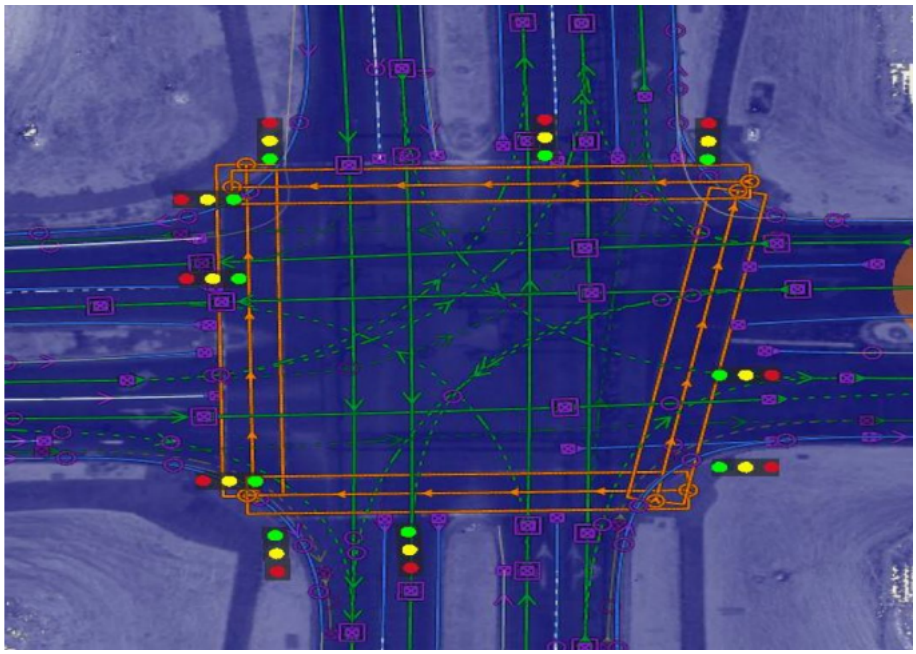


Figure 2.8: An intersection that has been mapped to provide important information for self driving cars in advance[7].

2.2.2 Autonomous Vehicles

Autonomous vehicles have in recent years become an area of high interest, spurring research and technological development[27][28][29]. The high interest in this technology is not surprising as there are great benefits to reap from a successful implementation of autonomous vehicles, such as making transportation more accessible and more efficient[30][31]. Driving is a complex procedure and consists of taking in a large amount of information while simultaneously handling a moving vehicle in real-time. It demands an ability to adjust to changing circumstances and sometimes also predicting what will be required next. In other words, it is a great challenge to create a system capable of performing on the same level as a human driver. The technology has seen great strides being made since the first modern implementations started appearing in the 1980's. As of March 2016, Google Inc. announced that their fleet of self-driving cars had traveled over 2,410,000 km autonomously, equal to approximately sixty laps around the equator[7]. Arguments are often made that self-driving cars actually can be safer than those controlled by humans, since a computer will not get tired or otherwise distracted[32].

Though the public availability of autonomous vehicles seem more a matter of time than anything else, there are still several hurdles that needs to be overcome before the technology can be considered ready, not all of which are technical. Several ethical aspect must be addressed as well, such as who will be liable in case the system

causes an accident. In addtion to that, how a system should act and prioritize if it is forced to choose between putting either its passengers or other road-users at risk also needs to be determined[33][34][35][36]. This is necessary since no autonomous vehicle system ever can be truly considered 100 % safe, but on the other hand, neither can human drivers.

2.3 Detection of Alzheimer's Disease from MRI images

One of the most common causes of dementia is Alzheimer's disease (AD). First described in 1906 by Alois Alzheimer, it has since then become a very common disease in the older population, afflicting as many as one out of every eight people above the age of 65, and nearly half the people over age 85[37]. Globally it is believed to afflict 35 million people. The world population is ever growing, and combined with longer life-expectancy, it is has been predicted by the World Health Organization that over 100 million people be will be suffering from AD by the year 2050[38].

AD is most easily observable as a gradual loss in cognitive functions with symptoms such as memory loss, confusion, irritability and sensitivity to stress being very common. Trouble with language and mobility have also been observed. Apart from these behavioral symptoms AD, can also be detected with changes to proteins in brain cells and structural changes in the patient's brain[37]. The structural changes can be observed using Magnetic resonance imaging (MRI). MRI is commonly used in medicine since it can produce images of the inside of the body. Observable changes due to AD can be atrophy of brain and inflated ventricles[39].

2.4 Libraries

In order to simplify the process of implementing machine learning algorithms, several software libraries have been developed. These provide tools that automatically execute many of the tasks required to set up, among other things, a functional neural network. Though the libraries are based on the same theoretical machine learning models, they differ in their approach on how to implement them. Below are short summaries of some of the more commonly used libraries.

2.4.1 Theano

Theano is a Python library for mathematical expressions in Python, developed with the goal of facilitating research in deep learning[40]. With Theano you can define, optimize, and evaluate mathematical expressions efficiently, using multi-dimensional arrays. Its syntax is similar to NumPy and this combined with optimized native machine code makes Theano a powerful tool, especially when implementing machine learning algorithms.

Theano optimizes the choice of expressions before computation and can then translate it either to C++ or CUDA, depending on if the program will be run on the CPU or GPU. Bergstra et al. showed in 2010 that implementing common machine learning algorithms using Theano are between 1.6× and 7.5× faster than competitive alternatives when compiled to run on a CPU and from 6.5× up to even 44× faster when compiled for the GPU[41].

2.4.2 Lasagne

Lasagne is a Python library meant to simplify the process of building and training machine learning algorithms with Theano. However, Lasagne is imported alongside Theano (and NumPy) and is not meant as a substitute. Some simple Theano code will usually be used as well as Lasagne, which is primarily a helpful tool.

2.4.3 Keras

Just like Lasagne, Keras is a high-level wrapper which runs on top of Theano. Additionally, it is also able to run on top of Tensorflow. Unlike Lasagne, which always uses some Theano code, Keras will not show any of the underlying work. It is designed to minimise overhead, to allow for fast and easy prototyping of machine learning algorithms.

2.4.4 Tensorflow

Originally developed by Google as part of the Google Brain project, Tensorflow was made open source in late 2015. Tensorflow is a Python library and stands apart by being the only one of the major libraries developed from the ground up by a major corporation, while the others have their origin in the research community. Tensorflow has some integrated quality of life tools such as TensorBoard, which allows the user to easily produce graphs visualizing things such as learning rate, model weights, loss functions and more. Tensorflow is also the only library that can distribute the workload not just across GPUs on the same device but on several connected devices, which can be a major computational advantage.

2.4.5 Caffe

Caffe was created by Yangqing Jia during his Ph.D in U.C Berkeley and though it is a C++ library, it uses python as its API. Caffe is excellent at implementing feed-forward networks and it is actually possible to do this without writing any code. This allows for quick testing and tuning when using existing networks[42].

2.4.6 Torch

Torch originated from NYU and is written in C and Lua. It is used by, among others, Facebook, Twitter, and the DeepMind project to implement neural networks. Since

Lua is a high-level interface for C it optimizes the code and allows for things such as for-loops to run much faster when compared to Python[42].

2.5 Deep Learning and Choice of Hardware

Given the size and large number of parameters a deep learning algorithm can contain, training often takes a very long time and the choice of hardware is therefore of utmost importance. This section aims to shed some light on the options available and discusses CPU versus GPU.

2.5.1 Central Processing Unit

Traditionally, neural networks have been trained primarily on a computer's Central Processing Unit (CPU). The CPU is often labeled as the 'brain' of a computer and operates by sequentially performing calculations sent to it. This forms the basis of how a computer works and the faster a CPU can perform its calculations, the faster it will finish the tasks given to it. Sometimes a program has different tasks that can be computed independently of each other. In order to optimize the time it takes to finish all tasks many CPUs have multiple cores that can perform calculations in parallel. This allows for tasks to finish quicker since they will not have to wait for the availability of a single core.

2.5.2 Graphics Processing Units

A Graphics Processing Unit (GPU) is a specialized kind of processor that excels at parallel computing. CPUs in consumer computers usually have between one and four cores and high-end server CPUs can have upwards of sixteen cores. GPUs make these numbers pale in comparison as they boast thousands of cores in top of the line GPUs.

GPUs are slower at sequential operations when compared to their CPU counterparts, but shines when given tasks that can be executed in parallel. This is common in 3D graphics, where a 3D landscape stored in the memory must be projected onto a 2D image to be shown on a display. As the name suggests GPUs were, and still are, primarily meant to be used to display 3D graphics.

Modern CPUs often have a GPU integrated into them but it can also be a separate chip on its own, this is often called a dedicated GPU. When a GPU is integrated it shares the system memory with the CPU and is allocated a part of it to use. When it is dedicated it usually has its own memory which is faster than the system memory and thus speeds up the operations. Laptops commonly have integrated GPUs but there are also models where a separate GPU is used. Desktop computers usually have a separate GPU available as a graphics card that can be inserted and removed from the computer. Sometimes both an integrated GPU and separate GPU can be used by the same computer for power-saving features. Since a dedicated GPU has a higher power consumption than the integrated GPU,

the integrated GPU is used until the computer needs to perform more intensive calculations, then it switches to the dedicated GPU.

Since the operations required in training a deep learning algorithm can be made in parallel, GPUs have risen to become a a highly valuable tool as they make the training several times faster than by using CPU alone. This does however require the ability to program the GPU to run different code, and must be supported by the manufacturers. In the high-end graphics card market today, realistically only two major manufacturers exist – AMD and NVIDIA. Of these two, NVIDIA have invested a lot into its CUDA language, which is designed to allow code to be run on their GPUs. As an alternative it is possible to use OpenCL which is used for parallel computing on most common types of processors from many manufacturers. OpenCL is maintained by The Khronos Group which is a non-profit consortium who creates open standard application programming interfaces (API)[43]. While the general applicability of OpenCL is very desirable, CUDA has been around longer and has better support by machine learning software. NVIDIA cards have thus become by far the most commonly used graphics card when working with machine learning algorithms[44].

Given the great interest in recent years many actors are working with the goal to have a part of this emerging new market. Intel has been tweaking their server CPUs to perform better with machine learning[45]. Google has been developing a chip that will perform machine learning tasks energy efficiently[46]. AMD has even announced a new line of GPUs designed specifically for machine learning to be released in 2017, along with software tools to facilitate better performance[47]. This indicates that the choice of hardware suited for machine learning is expanding rapidly, and leaves it up to speculation what choice will be best in the future.

3
Experimental Setup

All experiments were conducted on a custom built desktop computer running Linux. The computer specifications can be found in table 3.1 and the versions of the software libraries used in table 3.2. Short descriptions of the machine learning specific software libraries used can be found in section 2.4. For more information on the impact the choice of hardware can have on machine learning projects, see section 2.5.

Table 3.1: Computer specifications for this project.

CPU	AMD Phenom II X6 1055T
RAM	12 GB DDR3
GPU	NVIDIA GeForce GTX 970 4 GB
SDD	Intel 330 Series 180 GB
Linux kernel	4.4.0-21-generic

Table 3.2: Software libraries for deep learning used in this thesis.

Python	2.7.12	**Theano**	0.9.0.dev2
Lasagne	0.2.dev1	**Keras**	1.2.0
Nvidia driver	367.48	**CUDA**	release 8.0, V8.0.44
cuDNN	5.1	—	

4

Traffic Sign Recognition

This section describes the work of the primary focus of this thesis, namely traffic sign recognition. It begins by describing the methods investigated in this thesis in section 4.1, which is then followed by a section containing the experimental results and performance evaluation, section 4.2. Finally, a short discussion of the results can be found in section 4.3.

4.1 Methods Investigated in this Thesis

The main focus throughout this thesis has been the problem of designing a system to correctly classify images of traffic signs. The outline for training and testing using the German Traffic Sign Recognition Benchmark dataset (GTSRB) is explained in this section. First, the details of training, validation and testing are explained, followed by a description of the characteristics of the dataset itself. Lastly, a short description of the implementation is given.

4.1.1 Training, Validation, and Testing

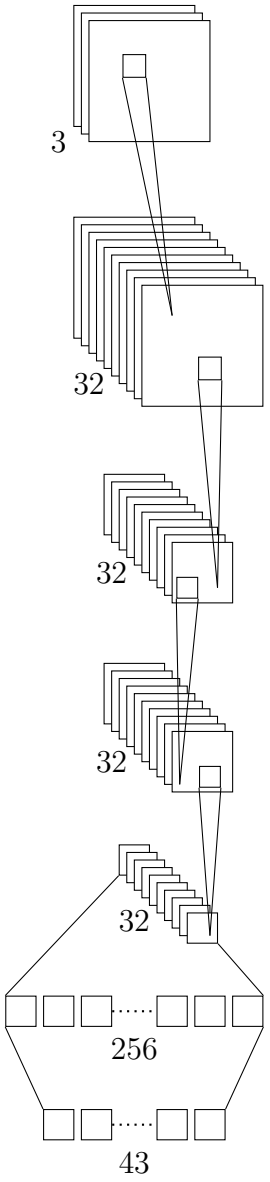
For the initial setup, a simple baseline CNN architecture was designed and a number of test cases constructed. The test cases can be seen as consisting of three parts. The first part was a systematic and quantitative setup, were one hyperparameter was alternated at a time, to test its impact on accuracy and run time of the training process. The second part consisted of taking the best results from the first part and combining them into an optimal network. Finally, for the third part the experiences from the quantitative testing was combined with recommendations found when studying previous work done, to try to come up with an even better system.

All test cases were run using 10-fold cross validation, i.e. the training set was split into ten equal sized parts and one part was used for validation, while the other nine were used for training. This was repeated ten times, ten folds, until each part had been used for validation, the validation accuracy then being the average accuracy from each fold. Each fold was run for ten epochs, one epoch being a full run through of the dataset. Their ability to generalise onto new datasets was then tested using the test set. The hyperparameters that generated the best results on the validation data were also tested one more time, but now running for 500 epochs instead of ten. This in order to see the results when the network was allowed to converge.

Two different data sets where used from the GTSRB, more thoroughly described in section 4.1.2, one large data set containing about 39,000 images used for training and validation, and one that was solely used for testing, containing about 10,000 images[48]. The two datasets are completely separate, meaning that if several pictures exists of the same sign from different distances and angles, they are not shared between the datasets.

The baseline architecture with the details of its design can be seen in figure 4.1. The input to the network consists of RGB images of size 32×32 pixels. The first layer is a convolutional layer with 32 filters of spatial size 5×5, applied with a stride of one and zero padding of two. This is followed by a rectifier that applies the function $f = max(0, x)$. The input has now been transformed to 32 feature maps of spatial size 32×32, and the color channels have been merged in the convolutional layer, meaning the feature maps can now only be viewed as grayscale, not RGB. Max pooling is then applied, using spatial filter size 2×2 and stride two (no padding), which halves the width and height of the feature maps.

After the pooling layer, another convolutional layer, rectifier, and additional pooling layer follows, all with the same respective hyperparameters as their first respective instance. The feature maps are now down to a spatial size of 8×8, while remaining 32 in number. Next comes two fully connected layers, both applying a dropout rate of 0.5 during training. The first applies another rectifier and outputs 256 units, the second applies the softmax function and outputs the 43 class probabilities.



Input: RGB image, size 32×32

Convolution: 32 filters, spatial size 5×5, zero padding 2, stride 1
ReLU: $f = max(0, x)$
Outputs 32 feature maps of spatial size 32×32

MaxPool: spatial size 2×2, no padding, stride 2
Outputs 32 feature maps of spatial size 16×16

Convolution: 32 filters, spatial size 5×5, zero padding 2, stride 1
ReLU: $f = max(0, x)$
Outputs 32 feature maps of spatial size 16×16

MaxPool: spatial size 2×2, no padding, stride 2
Outputs 32 feature maps of spatial size 8×8

FC: 0.5 dropout rate
ReLU: $f = max(0, x)$
Outputs 256 units

FC: 0.5 dropout rate, softmax
Outputs 43 units (class probabilities)

Figure 4.1: Initial baseline architecture used. Hyperparameters were changed one at a time during the quantitative testing in order to determine how they affect the accuracy.

For the quantitative test cases, the baseline architecture was used as the foundation and different hyperparameters were altered during each test case, to examine the impact of it on accuracy and run time. The constructed test cases are listed below:

Number of epochs The number of epochs for which the network was trained was alternated, from just once up to 2000 times. This was done to determine a reasonable amount of epochs for the network to converge properly. One epoch is one complete run through of the dataset.

Baseline architecture: Ten

Number of filters The number of filters in the convolutional layers were alternated, from 2 up to 512, using powers of 2.
Baseline architecture: 32 *filters*

Dropout rate Different dropout rates from 0.0 to 0.9, using increments of 0.1.
Baseline architecture: 0.5

Spatial filter size Spatial filter size for the convolutional layers was changed to 3×3 and 7×7 and tested both with and without padding.
Baseline architecture: 5×5*, with padding*

Depth The impact of the depth was tested by adding convolutional layers, from using one up to twenty. This was done twice, once with two max pool layers and once with three. Adding more pooling layers would decrease the size of the images too much.
Baseline architecture: Two convolutional layers, two pooling layers

Gradient Non-zero gradient, 0.01 and 0.33, was applied on the rectifiers after the convolutional layers. For details on gradient, see equation (2.1) in section 2.1.3.1.
Baseline architecture: 0.0

Max pool Spatial filter size of the max pool layers was changed to 1×1, i.e. no down sampling, and 4×4.
Baseline architecturee: 2×2

Learning rate Learning rate was alternated from 0.001 to 1.0 in steps by a factor of 3.
Baseline architecture: 0.01

Batch size Different batch sizes were tested, from 1 up to 2000.
Baseline architecture: 500

Input image size Sizes 8×8, 16×16, 32×32, 64×64, and 128×128 was used.
Baseline architecture: 32×32

The best results from the quantitative testing of the hyperparameters were then used to modify the baseline architecture, setting the hyperparameters to the ones that produced the best results during the quantitative testing, this in order to optimise both the accuracy and run time. Only the validation accuracies were taken into consideration, and not the test accuracies.

Additional architectures were also created, based on experiences from the quantitative testing and research done for the background section, to evaluate other network designs. Due to time constraints these networks were initially only allowed to train for 100 epochs. Their designs are listed below:

Architecture 1 The structure of the network is the same as in the baseline architecture outlined in figure 4.1, with an additional convolutional layer stacked before each pooling layer. The number of epochs was increased to 100 per training fold, to allow the network enough time to converge.

Architecture 2 Exactly the same as Architecture 1, except for one more additional convolutional layer before each pooling layers, making it three convolutional layers stacked before the pooling layers.

Architecture 3 The same as Architecture 1 with two stacked convolutional layers before the pooling layers. The first two convolutional layers makes use of 32 filters and the last two has 64 filters. Image input size was increased to 43×43, dropout lowered to 0.3 before the fully connected layers.

Architecture 4 The same as architecture 2, with three stacked convolutional layers before each pooling layer. The difference being that the number of filters in the convolutional layers are increasing with the depth of the network. The first three stacked convolutional layers have 32, 64, and 128 filters respectively and the three convolutional layers before the second pooling layer each has 256 filters.

4.1.2 Dataset

The dataset used for traffic sign recognition was the German Traffic Sign Recognition Benchmark (GTSRB)[48]. The images all contain one traffic sign only. They have all been cropped in a way that leaves some background noise, but the sign is in the center and takes up the majority of the space, as can be seen in figure 4.2, which shows example images from each class in the dataset. The GTSRB dataset contains 43 different types of traffic signs, which have been assigned as different classes numbered 1-43. Figure 4.3 shows the distribution of the different classes, for both the training and test sets.

The images vary in size and shape, but the absolute majority are roughly square, as can be seen in figure 4.4, which shows the relationship between the sides of the images, as the ratio of the larger over the smaller side of each image. The distribution of image sizes can be seen in figure 4.5. If a square shape is assumed they have a median value of 43×43, and a mean value of 56×56 pixels. The median value is considered to be the most representative as a small number of images with a comparatively large size inflate the mean value.

The data consists of two parts, one for training and validation, and one for testing. Many images are actually photos of the same sign from different distances, so there is a correlation between the images used for training and validation. However, the training and test datasets do not share any images of the same signs and are completely separate. When the training set is loaded all images are loaded into memory in order of class. After this they are randomly shuffled, then split into ten different parts to be used for 10-fold cross-validation. After loading the dataset, the only pre-processing done was changing the pixel values from integer values in range $0 - 255$ to floating point precision with range $0 - 255/256$.

Figure 4.2: Examples of traffic sign images and their respective class numbers.

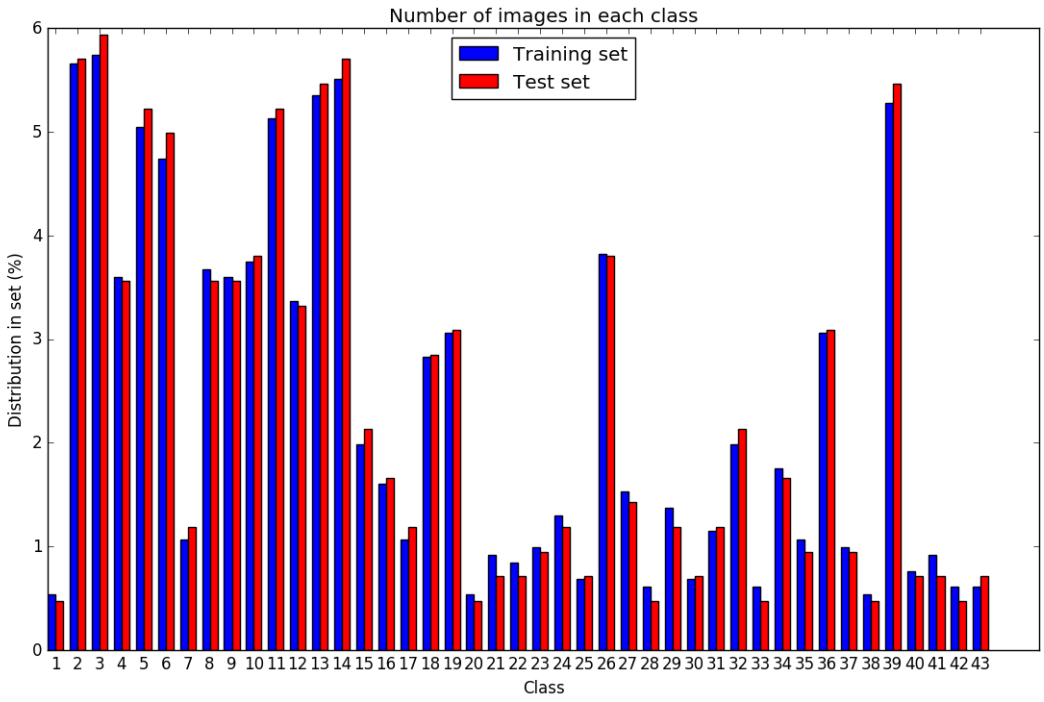


Figure 4.3: Percentage of the total number of images in the sets for each class.

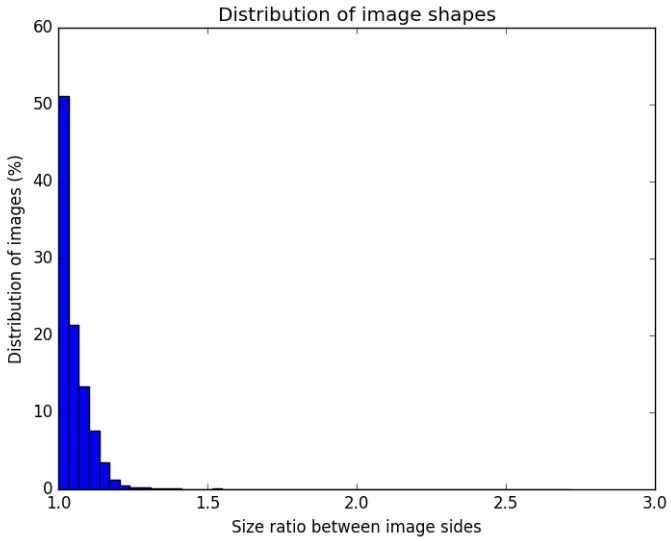


Figure 4.4: Distribution of image shapes in the GTSRB dataset, as the ratio of the longer over the shorter side of each image.

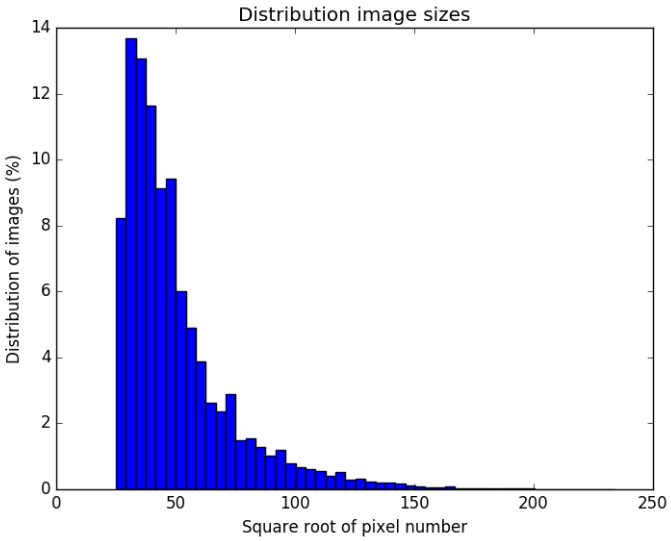


Figure 4.5: Distribution of image sizes in the GTSRB dataset. The x-axis shows the square root of the number of pixels in the images in order to better illustrate their approximate size, since the absolute majority of the images are roughly square as seen in figure 4.4. For example, when the x-axis shows 50 pixels, the images are assumed to be approximately 50×50 pixels in size.

4.1.3 Implementation

For the implementation, the example program `mnist.py` from the Lasagne repository on GitHub was used as a base for this project[49]. It was modified to be able to load the dataset, 10-fold cross-validation and additional functionality relevant to this project was implemented, including the design of the network. Cross-entropy was used as the loss expression to minimise during training, and the initial weights were randomly sampled from a uniform distribution.

4.2 Results and Performance Evaluation

The results from training and testing using the traffic sign dataset are presented here. Section 4.2.1 contains the results from the optimised networks and section 4.2.2 contains the quantitative test results.

4.2.1 Optimised Networks

In this section the test results and performance evaluation generated after training the optimised network structures, as described in section 4.1.1, are outlined. Section 4.2.1.1 contains the best results found in this study, which were found by combining the quantitative test results in section 4.2.2 with the results from the larger architecture designs in section 4.2.1.2.

4.2.1.1 Optimised Networks Based on Quantitative Test Results

The best results found in this study can be seen in table 4.1, the highest test accuracy achieved was 98.81 %, almost as good as human performance at 98.84 % on the same dataset[18]. The reason for the human performance not being 100 % is that some of the images are sometimes very small and blurry, making it difficult even for humans to visually recognise the signs correctly.

The first column in the table names the architecture, the details of which are described below. The second column contains the validation accuracy when running 10-fold cross validation, the third the test accuracy when using another dataset than the one used for training and validation, and finally the fourth column contains the training time for the network, not including the time for starting the program, loading images from the hard drive etc.

Additionally, the results of the best performing machine learning model of the International Joint Conference on Neural Networks, IJCNN, 2011 competition "The German Traffic Sign recognition Benchmark" has also been added for comparison in the table. It makes use of several deep CNNs, a committee, to classify the traffic signs correctly by averaging. The much more complex model took 37 hours to train on four GPUs, which type and model were not stated in the article, as compared to the models in table 4.1 which only took a few hours using just one GPU[50].

Table 4.1: Results for the optimised networks, based on results from quantitative testing in section 4.2.2 and larger architecture designs in section 4.2.1.2. The details about the architectures are listed below. 10-fold cross validation was used, with training for 500 epochs in each fold. The best results found were almost as good as human performance on the same dataset.

500 epochs			
Architecture	**Validation acc.**	**Test acc.**	**Training time**
Mod. Architecture 1	99.85 %	98.02 %	8 h 15 m 49 s
Ensemble mod. A1	–	**98.81 %**	–
Opt. baseline p	99.83 %	97.11 %	3 h 15 m 54 s
Opt. baseline no p	99.83 %	97.70 %	2 h 3 m 7 s
Ensemble opt. bc	–	98.38 %	–
Human performance[18]	–	**98.84 %**	–
Committee of CNNs[50]	–	99.46 %	–

The network that performed the best, an ensemble of the modified Architecture 1, is a combination of the larger architectural structures tested, results found in next section, and the optimal results found in the quantitative testing. Below are the network structures in table 4.1 described:

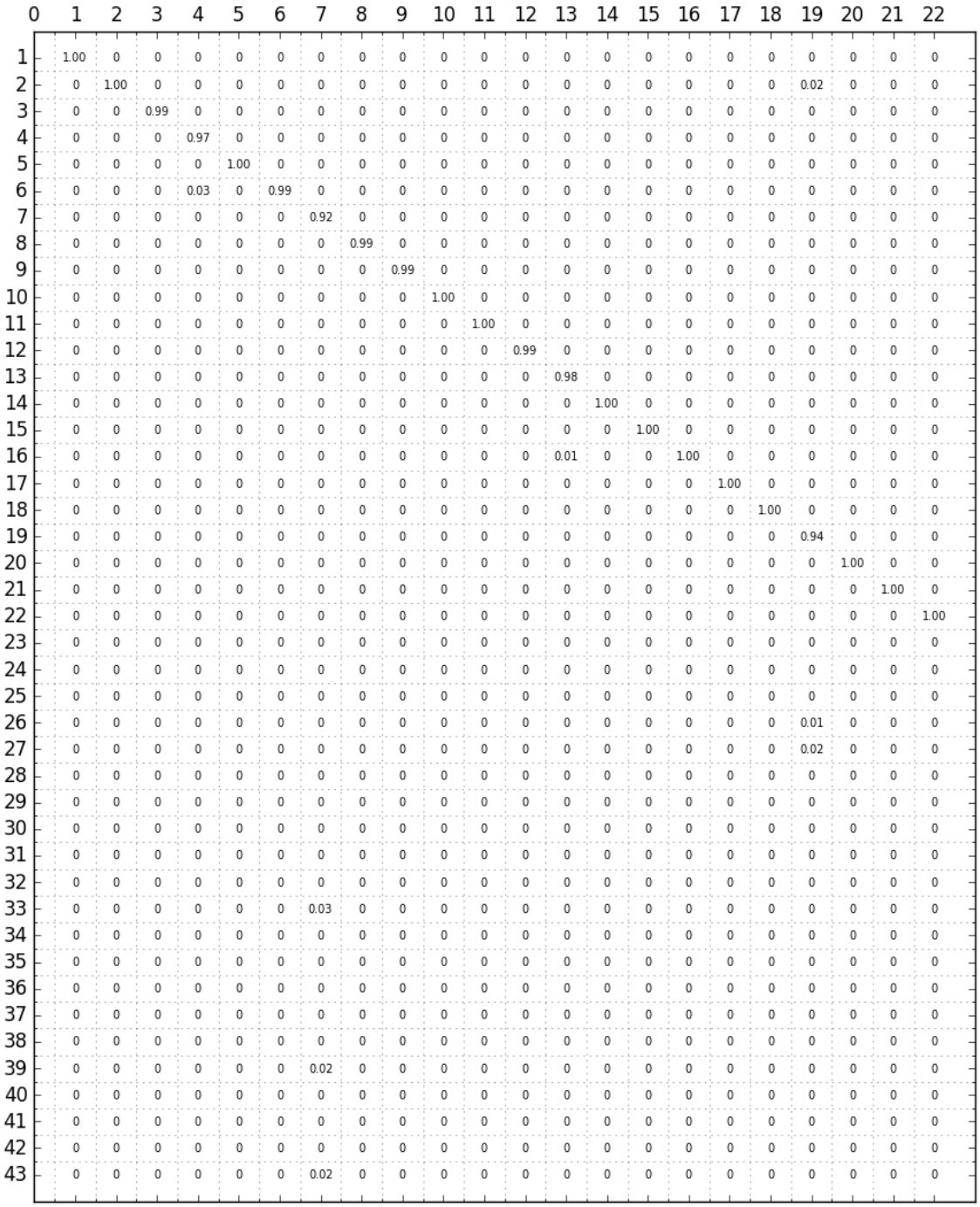
Mod. Architecture 1 Out of the larger architectures tested in section 4.2.1.2, when taking both run time and validation accuracy into consideration, Architecture 1 was the most suitable for further testing. The modifications were derived from the optimal settings found when doing the quantitative testing in section 4.2.2. The structure of the network is the same as in the baseline architecture outlined in figure 4.1, with an additional convolutional layer stacked before each pooling layer. The number of epochs was increased to 500 per training fold and learning rate was set to 0.03, additionally padding was only used for the second and fourth convolutional layer.

Ensemble mod. A1 The ten models trained during the 10-fold cross validation of the modified Architecture 1 were combined into an ensemble, to vote on what the signs should be classified as. This was the structure that generated the best results, with a test accuracy of 98.81 %, which can be compared with human performance at 98.84 %[18].

Opt. baseline p/no p The structure of the network was the same as for the baseline architecture in figure 4.1, with the addition of the optimal settings found from the quantitative testing. It was trained for 500 epochs in each fold, instead of ten, and the learning rate was increased from 0.01 to 0.03. It was tested both with and without padding for the convolutional layers. No other changes to the baseline architecture in the quantitative testing increased the accuracy and thus they were left unchanged.

Ensemble opt. bc Additionally, the ten trained models (one from each fold of the 10-fold cross validation) when not using padding were used to create an ensemble classifier. The ten models voted to classify the traffic signs, to try to further improve the results.

Figure 4.6 and 4.7 contain the confusion matrix for the ensemble classifier of the modified Architecture 1 in table 4.1. Each column represents the true class number, and each row the predicted class. In the matrix, the ratio with which the actual class in column j gets predicted as the class in row i can be found, between 0 and 1.00. Each column should sum to 1.00, the total probability. However, due to rounding of the rates, this is not always the case. The matrix only displays the results when sign j was classified as sign i at least 0.5 % of the times, otherwise it is shown as zero. If one sign is misclassified as several different signs but each only a few times, these values will not appear in the matrix, and therefore the sum may be slightly less than 1.00 in some instances.



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
1	1.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	1.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.02	0	0	0
3	0	0	0.99	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0.97	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	1.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0.03	0	0.99	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0.92	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0.99	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0.99	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	1.00	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	1.00	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0.99	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0.98	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.00	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.00	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	0	0	1.00	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.00	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.00	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.94	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.00	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.00	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.00
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.02	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0.03	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
39	0	0	0	0	0	0	0	0.02	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
43	0	0	0	0	0	0	0	0.02	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4.6: Part 1 of the confusion matrix generated by the ensemble classifier of the modified architecture 1 in table 4.1. This part shows what the classes numbered $1 - 22$ were classified as. The columns represent the actual class and the rows the predicted class, the value being the rate which the class given by the column is being predicted as the class given by the row. The misclassifications are rounded to nearest full percent, meaning only misclassifications above 0.5 % are shown. The second part of the matrix is found in figure 4.7.

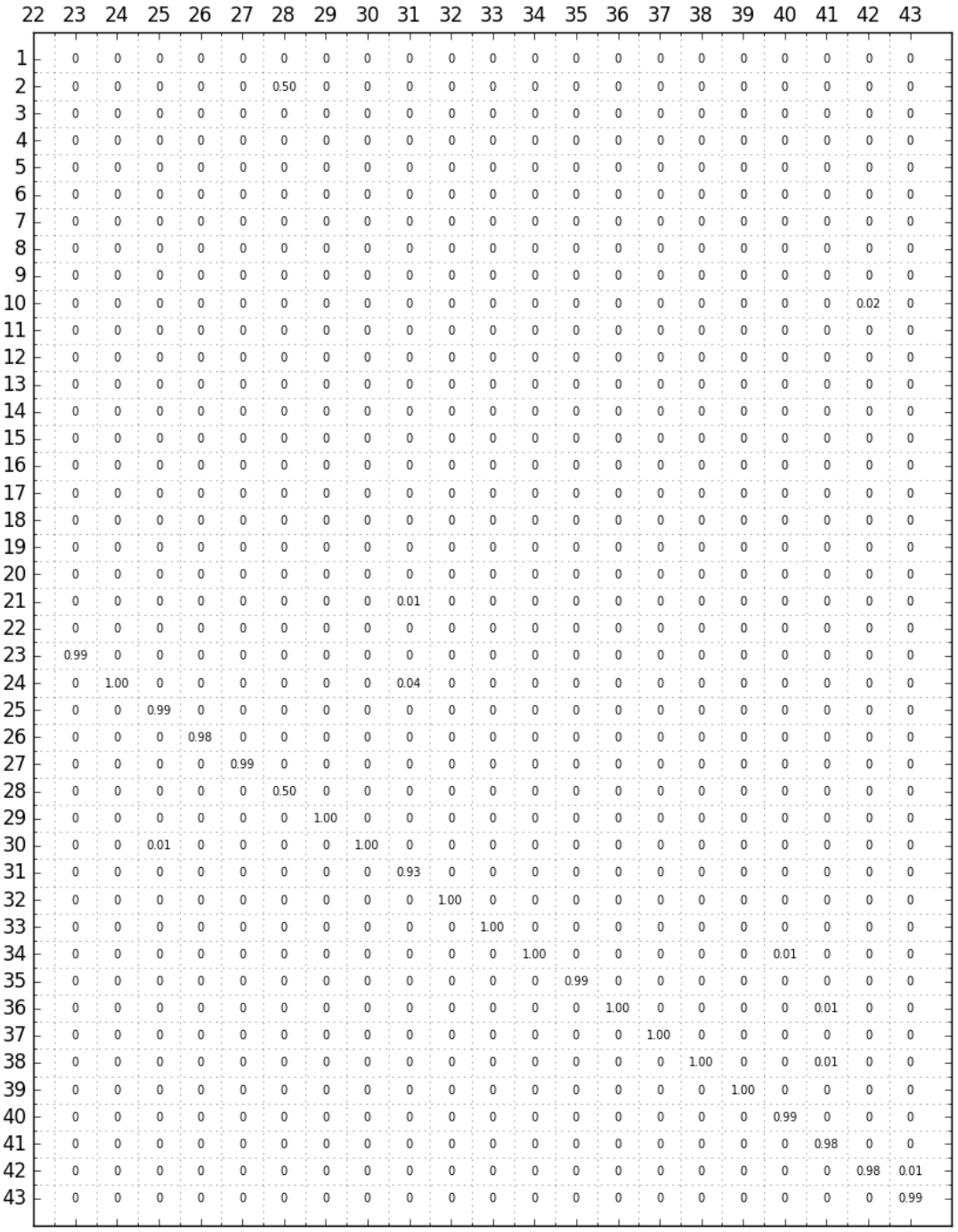


Figure 4.7: Part 2 of the confusion matrix generated by the ensemble classifier of the modified architecture 1 in table 4.1. This part shows what the classes numbered $23 - 43$ were classified as. The columns represent the actual class and the rows the predicted class, the value being the rate which the class given by the column is being predicted as the class given the row. The misclassifications are rounded to nearest full percent, meaning only misclassifications over 0.5 % are shown. The first part of the matrix is found in figure 4.6.

It is often difficult to understand just how a deep neural network attains its results. The features that the network learns to identify is far from always easy to discern for humans, which means errors can be difficult to explain since it is not immediately apparent how they arise. Perhaps the biggest issue with both CNNs and regular neural networks is that they are difficult to analyse, they are basically a black box with an input and an output.

Looking at the confusion matrix in figure 4.6 and 4.7 it can be seen that the network is better at identifying some classes than others. Eleven classes are sometimes misclassified as another class at least 0.5 % of the time or more. These classes are class 4, 7, 13, 19, 25, 28, 31, 40, 41, 42 and 43. Examples of how they look can be found in figure 4.8.

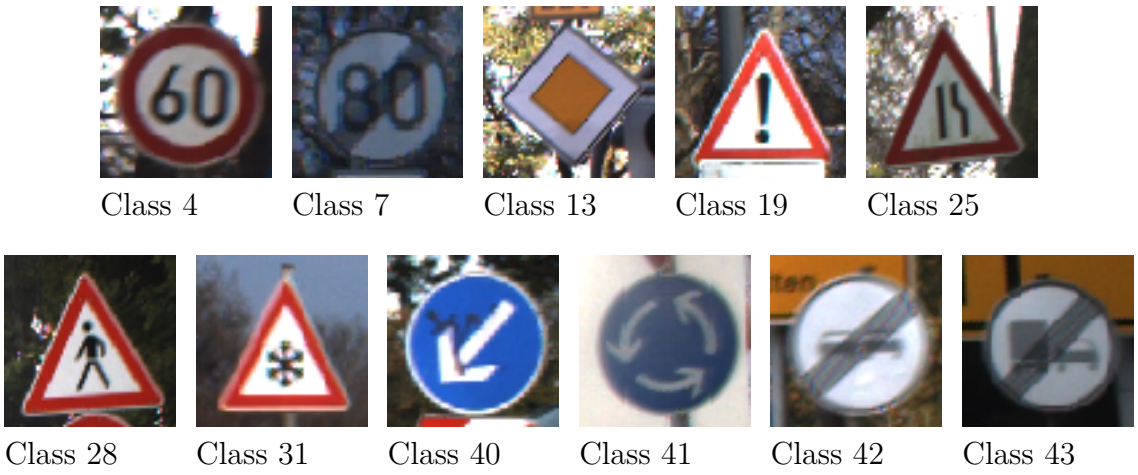


Figure 4.8: The eleven traffic signs that have misclassifications over 0.5%, for the modified architecture 1 ensemble classifier in table 4.1.

Class 4 is 3 % of the time misclassified as class 6, both can be seen in figure 4.9. This is quite easy to explain since the signs share the same shape and color scheme, as well as similar the center markings.

Actual class

Class 4

Misclassification

Class 6

Figure 4.9: Example of Class 4 and its most common misclassification.

Looking at class 7 with its 92 % accuracy, it is most commonly misclassified as classes 33, 39, and 43 at 3 %, 2 % and 2 % respectively, which can all be seen in figure 4.10. The signs represented by classes 33 and 43 share similar features with the misclassified sign. The shape and colors are identical while the markings are much the same, so it is easy to see how these could be mixed up. The last misclassification, class 39, is quite a bit harder to understand as the only clearly common feature is the round shape of the signs. Otherwise both colors and markings differ significantly.

Actual class

Class 7

Misclassifications

Class 33

Class 39

Class 43

Figure 4.10: Example of Class 7 and its three most common misclassifications.

Class 13 stands out in the dataset as it shares neither shape nor color scheme with any other class. Intuitively this makes one think that it would be easy to classify because of the unique features, but it is not necessarily how the CNNs works. Class 13 has 98 % accuracy, the majority, 1 %, consisting of class 16 which can be viewed

as somewhat similar, as both lack inner markings and are bordered, examples can be found in figure 4.11. It is possible that since the network has been trained with many more signs that share shapes and color schemes, it has become better at discerning those types of signs, and have not become good at detecting the unique features of class 13. This is of course just speculation since there is no way to really discern what features the networks pick out.

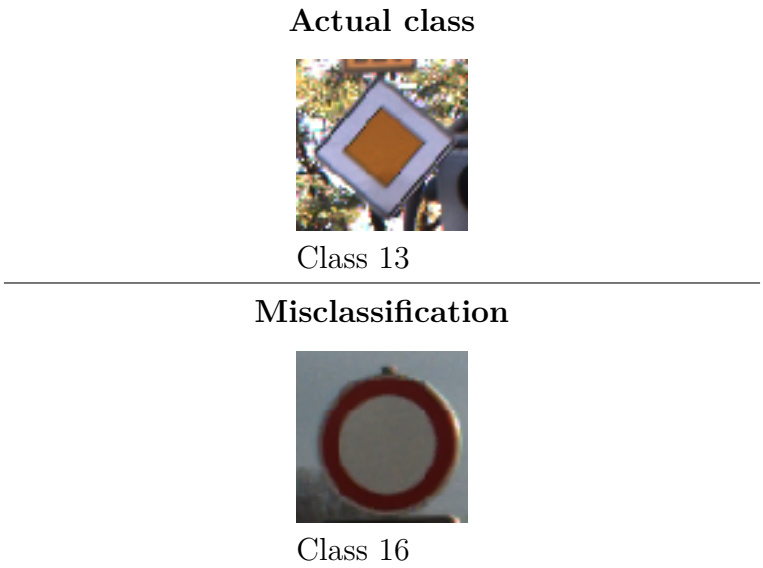


Figure 4.11: Example of Class 13 and its most common misclassification.

Class 19 is misclassified as classes 2, 26, and 27 at 2 %, 1 %, 2% respectively, they can be seen compared in figure 4.12. Classes 26 and 27 can be seen as quite similar when taking shape and colors into consideration. It is noteworthy that the center markings of class 27 are of a different color scheme, yet it is one of the more common misclassifications, why is unclear though. It is more difficult to understand why it is being misclassified as class 2, since while it shares the same color scheme, its shape is determinedly different. Additionally, it is not misclassified as any other speed limit sign, just this one.

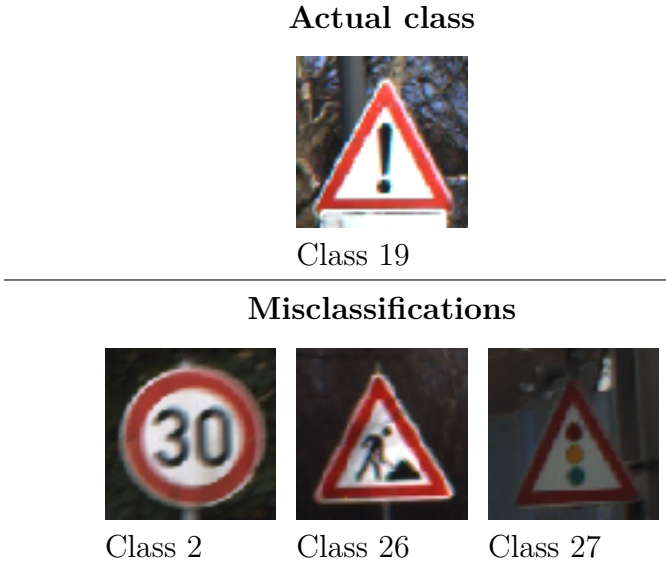


Figure 4.12: Example of Class 19 and its three most common misclassifications.

Class 25 is most commonly misclassified as class 30, this happens 1 % of the time, they can be seen in figure 4.13. Just why class 30 stands out and no other signs with the same shape is not immediately apparent as the center markings are quite dissimilar.

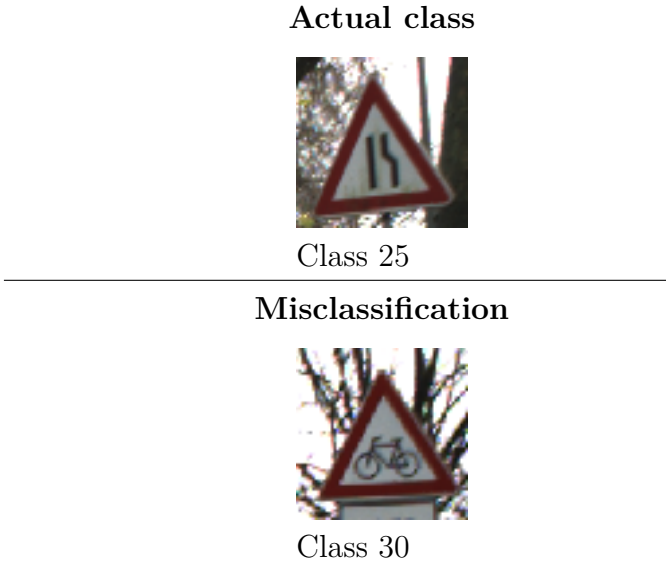


Figure 4.13: Example of Class 25 and its most common misclassification.

Class 28 stands out with the lowest accuracy at only 50 %. Virtually all of the misclassifications are as class 2. Example images of the classes can be seen in figure 4.14. Class 2 differs in shape and center markings from class 28 visually, but still it is obvious the network has trouble distinguishing between the two classes, as half the time it detects class 28 as class 2, it does however not have the same problem the other way around. It is also interesting to note that it is only misclassified as class 2, the 30 speed limit sign, and not other speed limit signs, that to a human eye

appear very similar to each other. No other class comes even near this low accuracy, and when looking further at the training and test dataset a possible explanation is found. When looking at class 28 in the test set half the images are of a sign that is partly covered by a shadow, this images was chosen as the example image in figure 4.14. The effect seems to somehow confuse the network causing it to classify the image incorrectly- This shows that a CNN cannot discern on its own which features are of importance to the subject at hand. It merely finds similarities between the images it is trained.

Actual class

Class 28

Misclassification

Class 2

Figure 4.14: Example of Class 28 and its most common misclassification.

Class 31 is most commonly misclassified as class 21 at 1 % and as class 24 at 3 %, the classes can be seen in figure 4.15. Again, these classes share similar features in shape and color. When looking at the confusion matrix in figure 4.7 it can be seen that the column values only add up to 0.98, i.e. 2 % of the classifications are "missing". This means that a few misclassifications have been distributed over other classes but make up less than 0.5 %. These are not displayed, since it is not known which classes they are.

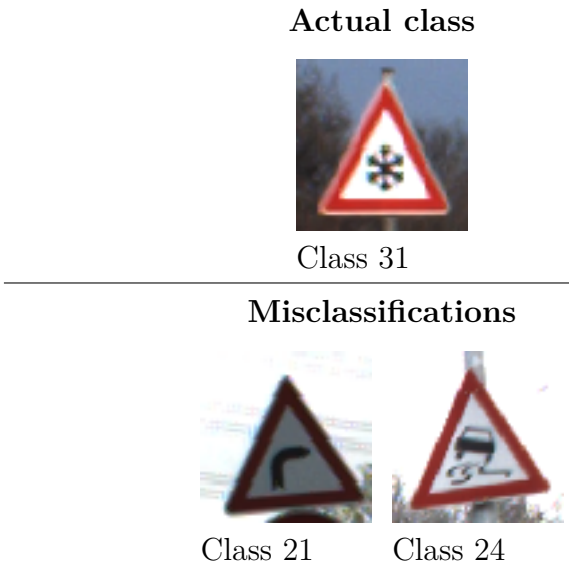


Figure 4.15: Example of Class 31 and its two most common misclassifications.

The most common misclassification of class 40 is class 34 at 1 %, examples of the classes can be seen in figure 4.16. The signs can be seen to share shape and color scheme. The markings are also somewhat similar, though they differ in orientation.

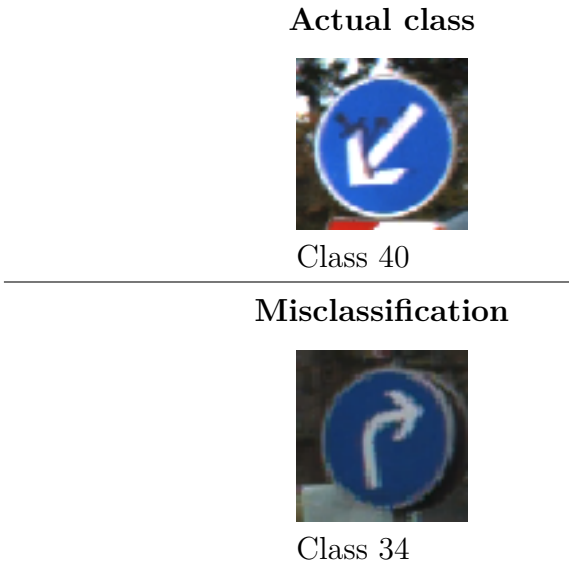


Figure 4.16: Example of Class 40 and its most common misclassification.

Class 41 is misclassified as class 36 and class 38, they each make up 1 % of the cases, examples of the classes can be seen in figure 4.17. Class 40 shares shape and color schemes with its misclassifications. The orientation, shape, and number of markings differ quite a bit, but they all show arrows, so it is possible the network cannot properly discern different arrows and their orientations properly.

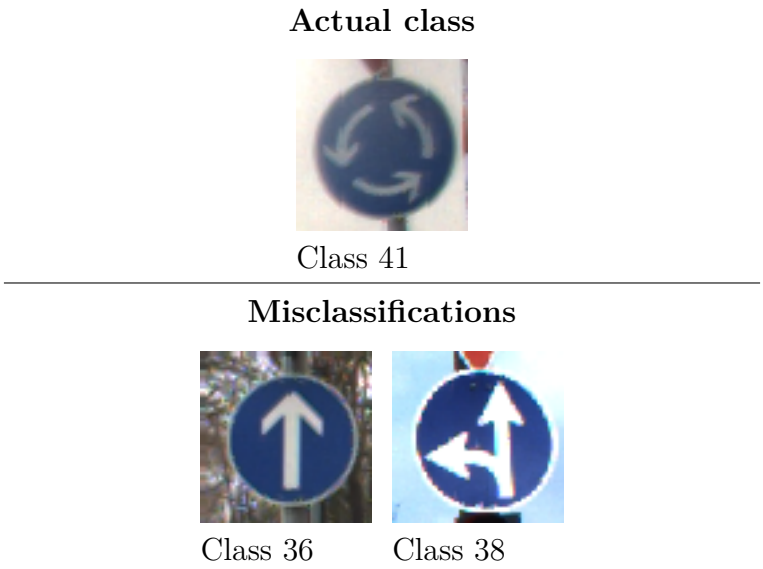


Figure 4.17: Example of Class 41 and its two most common misclassifications.

The most common misclassification of class 42 is class 10, this happens in 2 % of the instances. The classes can be seen in figure 4.18. Though they do not share the same color scheme, the shape is the same and the center markings very similar. This is the likely reason why the network has them mixed up.

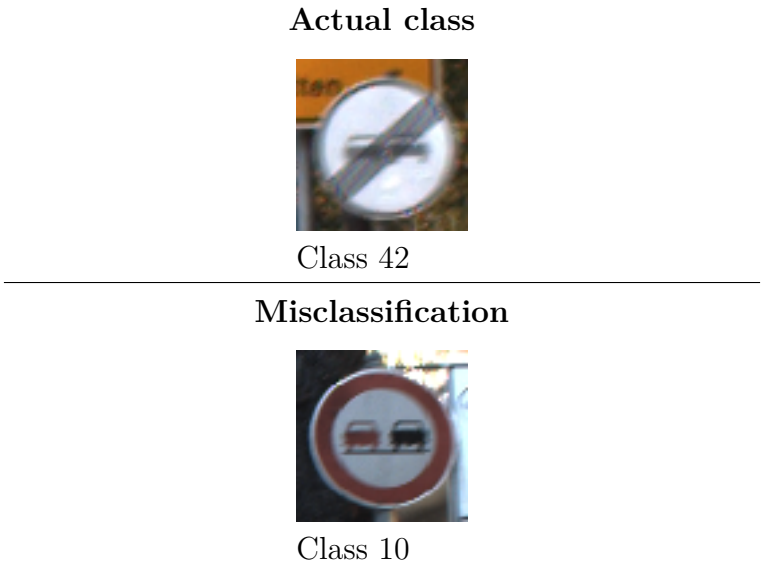


Figure 4.18: Example of Class 42 with its misclassification.

Finally Class 43 is 1 % of the instances misclassified as class 42, examples of the classes can be seen in figure 4.19. The signs are very similar with the only difference being part of the inner markings. This makes it easy to see how the network could mix them up.

Actual class

Class 43

Misclassification

Class 42

Figure 4.19: Example of Class 43 and its most common misclassification.

As could be seen from this section, it is not always evident why a sign is misclassified as another. It is virtually impossible to analyse how a CNN makes its decisions, which is one of its drawbacks, but it definitely does not yet see things the same way as we humans do. Noise in the images, such as shadows and obstructing objects, can hinder the classification. This is particularly apparent when the dataset contains several images of the same sign, just from slightly different angles, and not always many truly different versions of the sign, which was the case for the dataset used in this study and a likely cause for some of the misclassified instances.

4.2.1.2 Additional Architectures Tested

In this section, the results from testing on the additional network designs are presented, i.e. those derived from experiences from the quantitative testing in combination with research done prior to this study, which resulted in section 2.1 in the background section. The belief was that these larger network designs would generate better results, details about the architectures can be found in section 4.1.1, which are presented in table 4.2. The first column names the architecture, per definitions in section 4.1.1, the second the validation accuracy generated by 10-fold cross validation, the third the test accuracy on the separate test set, and the fourth column the training time. Training was run for 100 epochs in each fold.

When comparing the results of Architecture 1 in table 4.2 to the results of the baseline architecture with 100 epochs in the table in figure 4.21 there is an increase in test accuracy of 1 %. This is interpreted to mean that the extra convolutional layers provide better accuracy, at least for this amount of training.

Architecture 2 stacks even more convolutional layers, but here it appears that a limit has been reached in the increase of accuracy. Test accuracy is only marginally higher when compared to Architecture 1. This suggests that extra convolutional layers no longer extract any better information for the network to process and the added complexity is unnecessary.

Table 4.2: Results for the optimised networks, details in section 4.1.1. 10-fold cross validation was used, each network was trained for 100 epochs during each fold.

100 epochs

Architecture	Validation acc.	Test acc.	Training time
1	99.70 %	96.70 %	2 h 19 m 3 s
2	99.70 %	96.94 %	3 h 45 m 41 s
3	99.68 %	95.95 %	5 h 45 m 25 s
4	99.73 %	95.91 %	32 h 43 m 16 s

The results of Architecture 3 performs slightly worse than Architecture 1. The image size used has been increased to match the median image size but does not lead to better performance when compared to Architecture 1. This can be taken to mean that increased image size simply leads to faster convergence and thus matters less when using more epochs. Architecure 3 also uses more filters in the later convolutional layers, which when looking at the table in figure 4.22 appears to increase accuracy. This seems to indicate that it may only converge faster, and not in fact generate the better accuracy that was observed earlier. The worse results can also be due to the lowered dropout rate, it is difficult to tell exactly what generated these results when several hyperparameters were changed simultaneously.

Architecture 4 was designed to see if an even larger network would produce a better accuracy. In practice this was not the case and in the end it only took much more resources to produce similar results. The total run time when training the network was over 33 hours, which was the result of using both more layers and filters. Training this network using 500 epochs instead of 100 epochs, as was done for the best performing networks, would increase the training time to over a week. A vastly increased number of filters would in theory give the network more ability to adjust and provide more accurate results. However, there is also always the possibility of a more complex model introducing overfitting.

4.2.2 Quantitative Test Results

This section displays the results when varying the different hyperparameters of the baseline architecture, as described in section 4.1.1. This part of the thesis was undertaken primarily to examine the impact of the hyperparameters on the accuracy and run time of a CNN.

4.2.2.1 Initial Setup and Baseline Architecture

The test results on the baseline architecture can be seen in table 4.3. The baseline architecture, with its initial settings, was chosen for its simplicity and fast training, to be able to generate a lot of test data in a short amount of time. It is not meant to be used as a traffic sign classifier, but rather as a benchmark to investigate the impact of changing the settings of the various hyperparameters.
Since a CNN with self learned features alters its input when it travels through the layer it gets progressively harder to discern what the features it uses actually are.

Table 4.3: Validation and test accuracies of the baseline case, described in section 4.1.1, in addition to training time for running 10-fold cross validation on the training data set. Each fold is run for ten epochs.

Architecture	Validation acc.	Test acc.	Training time
Baseline architecture	90.44 %	83.48 %	3 m 55 s

In figure 4.20 the feature maps obtained after sending a picture through the first convolutional layer in the baseline architecture is shown. It can be seen that even though this is the first layer, the actual features found are not readily apparent.

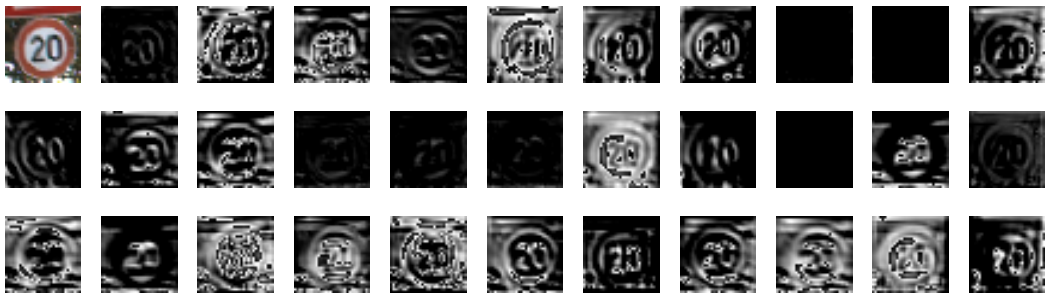


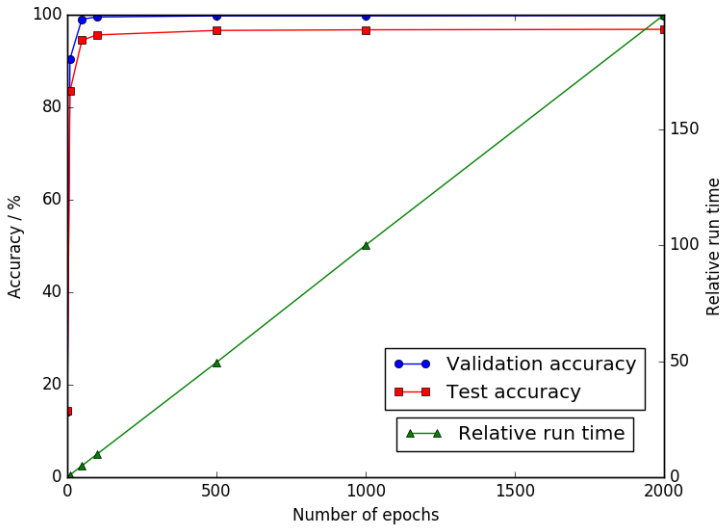
Figure 4.20: Feature maps obtained after the image in the top left has been run through the first convolutional layer in a trained baseline architecture network.

The figures and tables that display the results are all structured in the same way. The first column describes the changes compared to the baseline architecture, e.g. in the table in figure 4.21 the number of epochs are changed while all other hyperparameters are kept the same, therefore the first column contains the number of epochs used for each test. The second column shows the validation accuracy on the validation data set using 10-fold cross validation during training, and the third column the results after testing on the previously unseen test data set, both display the accuracy as ratio of correctly predicted images in percentage. The fourth and final column contains the training time, meaning only the time it takes to train the network, not any additional time spent on, for instance, starting the program or loading input into memory. Additionally, some tables have been visualised graphically, by displaying the validation and test accuracy as functions of the alternated hyperparameter for the specific test instance, these graphs also show how the time to train a network varies.

4.2.2.2 Epochs

The impact of changing the number of epochs used in each fold during training can be found in figure 4.21, note that the instance with 10 epochs marked in bold is the same as the baseline architecture in table 4.3. For one epoch, the fold accuracy varied with almost ten percentage units. This decreased to almost four percentage units when running for ten epochs and for fifty or more epochs, the fold accuracies were almost constant. Not surprisingly, with more epochs the accuracy increases. Interestingly however, is the fact that already somewhere between ten and fifty

epochs, the accuracy starts to converge. Training time appears to increase linearly with the number of epochs, which makes sense since the run time for an epoch is mostly constant. This means there is a trade-off between training time and accuracy.



Epochs	Validation acc.	Test acc.	Training time
1	14.27 %	14.40 %	24 s
10	**90.44 %**	**83.48 %**	**3 m 55 s**
50	99.05 %	94.48 %	19 m 19 s
100	99.53 %	95.65 %	38 m 37 s
500	99.77 %	96.62 %	3 h 13 m 29 s
1000	99.78 %	96.76 %	6 h 31 m 33 s
2000	99.85 %	96.86 %	13 h 0 m 45 s

Figure 4.21: The impact on accuracy and training time by varying the number of epochs in the baseline architecture. The table shows the absolute time values with the baseline case of ten epochs shown in bold. In the graph the training times are shown as time relative to the baseline architecture, 235 seconds, for easy comparison.

Ten epochs, which was the initial setting for all test cases, was too short an amount of time for the network to converge and often the accuracy would vary with up to a few percentage units between each fold, e.g. for the baseline architecture validation it varied between 88.60 % and 92.23 %. Only the average accuracy over all folds are shown for easier comparison between the different cases. For some test cases, the accuracies would vary even more, for example when using only two filters the accuracy would vary between 5.5 % and 60.5 % for the separate folds. This is important to take into consideration when looking at the results, which should be viewed as a guide to what happens early in the training.

Longer runs, i.e. more epochs, could show that what appears like poor performance early on in the training for some set of hyperparameters might actually converge to a higher accuracy in the end, if allowed to train for long enough. Therefore, the values of the hyperparameters tested that were considered best when run for ten epochs, were also tested using 500 hundred epochs, to see its possible impact

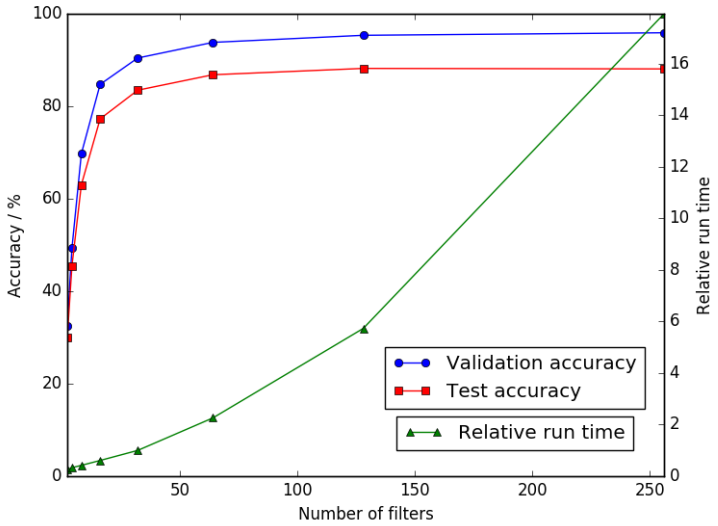
on accuracy when allowed more time to converge. 500 epochs was chosen since it had a good balance between the attained accuracy and the time it took to train the network. It was also the amount of epochs used in the optimized network whose results can be viewed in table 4.1

4.2.2.3 Number of Filters in Convolutional Layers

The results of changing the number of filters can be seen in figure 4.22. The accuracies for each fold when using just two filters varied greatly, from 5.5 % up to 60.5 %, this variation decreased as the number of filters increased. Four filters generated a variance in accuracy from 20.17 % to 67.54 % and 256 filters a very low variance, from 95.17 % to 96.63 %.

As can be seen seen in the table in figure 4.22, increasing the number of filters in each convolutional layer seems to increase the accuracy on both the training data and also unseen test data when training using ten epochs. However, when increasing the number of epochs the impact of increasing the number of filters seems to have virtually no impact in accuracy. Interestingly, the test accuracy even decreases a little. Why is unclear, but it suggests that a very complex network is not necessary to solve the problem at hand.

It can also be observed that the time it took to train the network increases exponentially. The run time is very dependant on the hardware used. Since a relatively powerful GPU was used in this study, it made it possible to efficiently run the computations in parallel, because they are independent of each other. But with more added filters there are more computations and eventually it will not be able to parallelize all of them optimally. This is speculated to be the reason why the run time does not increase linearly with more filters. The results are not surprising, with more filters more information or features can be extracted from the input and yield more accurate results. These tests also show a limitation of the test setup, as in the case when using 512 filters. When the amount of data required to train the dataset for one epoch does not fit into the graphics memory and the program cannot run. This can however be alleviated by using smaller batches in order to lower the size of data handled during one epoch.



10 epochs

No. of filters	Validation acc.	Test acc.	Training time
2	32.58 %	29.95 %	1 m 6 s
4	49.33 %	45.50 %	1 m 18 s
8	69.74 %	63.01 %	1 m 38 s
16	84.74 %	77.28 %	2 m 23 s
32	**90.44 %**	**83.48 %**	**3 m 55 s**
64	93.82 %	86.82 %	8 m 51 s
128	95.35 %	88.19 %	22 m 27 s
256	95.90 %	88.08 %	1 h 10 m 14 s
512	Out of memory		

500 epochs

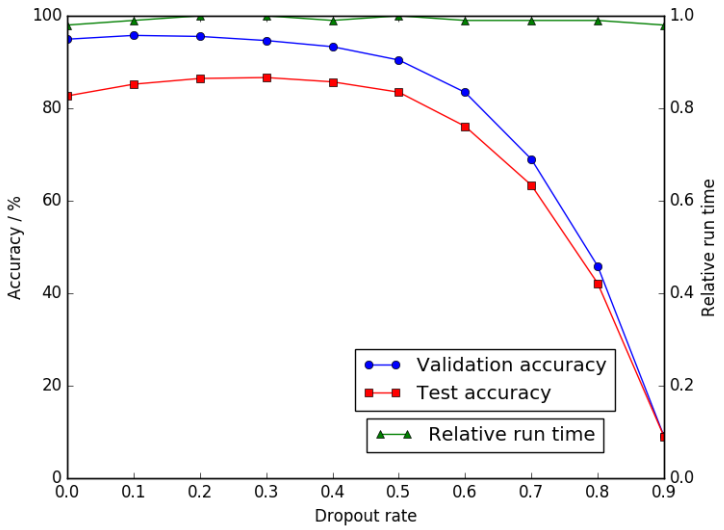
No. of filters	Validation acc.	Test acc.	Training time
32	99.77 %	96.62 %	3 h 13 m 29 s
128	99.78%	96.05%	18 h 39 m 31 s

Figure 4.22: Validation and test accuracy, along with training time, when alternating the number of filters in the baseline architecture, which is displayed in bold in the top table. The graph visualises the top table, training time is here displayed as relative to the baseline architecture, 235 seconds. 10-fold cross validation was used and each fold was run for ten epochs. The bottom table shows the results when running with 500 epochs instead of ten.

4.2.2.4 Dropout Rate

Figure 4.23 contains a table and graphic visualisation of the results when changing the dropout rate on the last two fully connected layers in the baseline architecture. The dropout rate always is kept the same for both layers in these tests. Note that 0.5 dropout rate is the baseline architecture. The accuracy of each fold varied with just 1-2 % for a dropout rate of 0.4 or less, and then it started to slowly increase. In

this implementation, due to how the dropout layer is implemented in Lasagne, the dropout rate is given as the probability a node will be deactivated during training.



10 epochs

Dropout rate	Validation acc.	Test acc.	Training time
0.0	94.94 %	82.68 %	3 m 50 s
0.1	95.76 %	85.21 %	3 m 53 s
0.2	95.54 %	86.45 %	3 m 55 s
0.3	94.65 %	86.65 %	3 m 55 s
0.4	93.32 %	85.72 %	3 m 53 s
0.5	**90.44 %**	**83.48 %**	**3 m 55 s**
0.6	83.45 %	76.10 %	3 m 53 s
0.7	68.98 %	63.37 %	3 m 53 s
0.8	45.86 %	42.14 %	3 m 53 s
0.9	9.07 %	9.02 %	3 m 50 s

500 epochs

Dropout rate	Validation acc.	Test acc.	Training time
0.3	99.68 %	94.77 %	3 h 15 m 30 s
0.5	99.77 %	96.62 %	3 h 13 m 29 s

Figure 4.23: Validation and test accuracy, along with training time, when running the tests with different values of the dropout rate. The dropout is applied to the two fully connected layers at the end of the network, and denotes the probability with which each node is deactivated during training. Dropout rate 0.5 was used in the baseline architecture, which can be seen in the top table, which contains the results when running each fold in the 10-fold cross validation for ten epochs. The bottom table contains the results when increasing the number of epochs to 500. The graph visualises the results from the top table.

The extremely low results for a high dropout rate, as can be seen in the table in figure 4.23 are expected, if the nodes are discarded during training 90 % of the time,

they will almost never get a chance to learn. A dropout rate of 0.5 is often suggested and used, although the results from this study when training for ten epochs suggests that perhaps 0.2 or 0.3 could potentially be better. However, when increasing the number of epochs to 500 the more commonly used dropout rate of 0.5 outperforms the lower dropout rate of 0.3. The training time is virtually unaffected by different dropout settings. This makes sense since dropout is only used in the fully connected layers. The convolutional layers require more computations and the gains from using a higher dropout rate are therefore not measurable.

4.2.2.5 Spatial Filter Size and Zero Padding

Removing the zero padding of the two convolutional layers in the baseline architecture generated the results found in table 4.4. Padding of the baseline architecture was two and fold accuracy varied with just a couple of percentage units. The models were trained for both ten and 500 epochs.

Table 4.4: Test results when varying the zero padding in the convolutional layers of the baseline architecture, using 32 filters of spatial size 5×5. Padding of the baseline architecture is two. The top table displays the results when running each fold for ten epochs in the 10 fold-cross validation, the bottom when the number of epochs are increased to 500.

10 epochs

Padding	Validation acc.	Test acc.	Training time
0	90.54 %	85.30 %	2 m 26 s
2	**90.44** %	**83.48** %	**3 m 55 s**

500 epochs

Padding	Validation acc.	Test acc.	Training time
0	99.76 %	97.49 %	1 h 59 m 42 s
2	99.77 %	96.62 %	3 h 13 m 29 s

Not using padding did not have much impact on the accuracy during these tests, but more on the training time due to it decreasing the size of the images as they move through the network, leading to less computations. Padding is more important for deeper networks, so as not to lose too much information with the convolutions. For just two convolutional layers, as was used here, the pixels on the borders are mostly part of the background and do therefore not contain any important information to correctly classify the signs.

To test the effects of the filter size the spatial filter size in the convolutional layers was changed to 3×3 and was tested both with and without padding, the results of which can be found in table 4.5. The same was also done for a spatial filter size of 7×7, as can be seen in table 4.6. The tests were run for both ten epochs in order to see what happens early in the training during training, and for 500 epochs in order to compare the final accuracy when the networks have been allowed to fully converge. The padding was chosen so as to keep the size of the input the same, one and three for the 3×3 and 7×7 filters respectively. When training the networks

for 10 epochs the fold accuracy varied less when using 7×7 filters, just 1-2 %, than when using filters of size 3×3, which varied with almost 10 % for the separate folds. The results were similar both with and without padding.

Table 4.5: Results when using filters of spatial size 3×3 for the convolutional layers, both with and without padding. The top table displays the results when running each fold in the 10-fold cross validation for ten epochs, the bottom when the number of epochs are increased to 500.

10 epochs

Padding	Validation acc.	Test acc.	Training time
0	83.22 %	75.78 %	2 m 14 s
1	81.62 %	72.96 %	2 m 37 s

500 epochs

Padding	Validation acc.	Test acc.	Training time
0	99.71 %	96.71 %	1 h 54 m 48 s
1	99.73 %	95.30 %	2 h 11 m 53 s

Table 4.6: Results when using filters of spatial size 7×7 for the convolutional layers, both with and without padding. The top table displays the results when running each fold in the 10-fold cross validation for ten epochs, the bottom when the number of epochs are increased to 500.

10 epochs

Padding	Validation acc.	Test acc.	Training time
0	90.64 %	86.63 %	2 m 21 s
3	92.19 %	85.42 %	6 m 44 s

500 epochs

Padding	Validation acc.	Test acc.	Training time
0	99.67 %	97.32 %	1 h 58 m 12 s
3	99.77 %	96.34 %	5 h 14 m 44 s

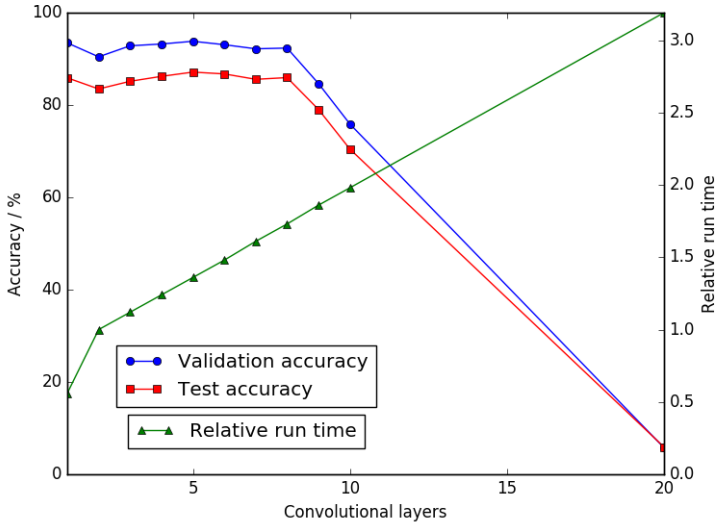
Varying the spatial filter size of the convolutional layers appears to have some impact. The baseline architecture used a spatial filters size of 5×5, table 4.5 and table 4.6 contain the results of using spatial filter size 3×3 and 7×7 respectively. Decreasing the size generates a slightly worse performance, independent of using padding or not, whereas increasing the filter size generates performance comparable to the baseline architecture, but not better. Filter size also affects the run time since smaller filters means fewer computations than larger filters.

4.2.2.6 Depth of Network

Figure 4.24 displays the results when increasing the depth of the network. The structure was kept the same, but for each added convolutional layer a ReLU and a

MaxPool layer was also added, up to and including depth two. After that additional layer, no more MaxPool layers where added to keep the minimum size of the images at 8×8. Similarly, in figure 4.25, the results can be seen when adding a MaxPool layer also after the third convolutional layer, decreasing the minimum image size to 4×4. The fold accuracy varied more with deeper networks, with just a few percentage units for the shallower networks. At nine layers for the network in table 4.24 and six layers for the network in table 4.25, the fold accuracies started to vary greatly with more than 10 % for each fold. Using twenty layers gave the same poor results for each fold.

The tables in figures 4.24 and 4.25, contain the results of changing the number of convolutional layers, with two and three max pooling layers used respectively. When using just two pooling layers, the results are similar up to a depth of around eight convolutional layers, and then the accuracy starts to decrease drastically. When using three pooling layers, the results almost immediately starts to decrease. The training time increases linearly in both cases since adding more layers means a set amount of extra computations. The slope is less steep when using three pooling layers, this is assumed to be due to the smaller size of the feature maps. The poor performance for deeper layers does not necessarily mean that they are objectively worse, just that they perform worse with these settings. The reason for the poor performance when going deeper is probably that ten epochs are too few to train the network, since with an increased depth more trainable weights are added and therefore increasing the complexity of the network. This was tested by running the networks for 500 epochs with depths 3, 5, and 10, greatly increasing the accuracy, which was expected. It did however not yield any better results when compared to the baseline architecture using just 2 convolutional layers.



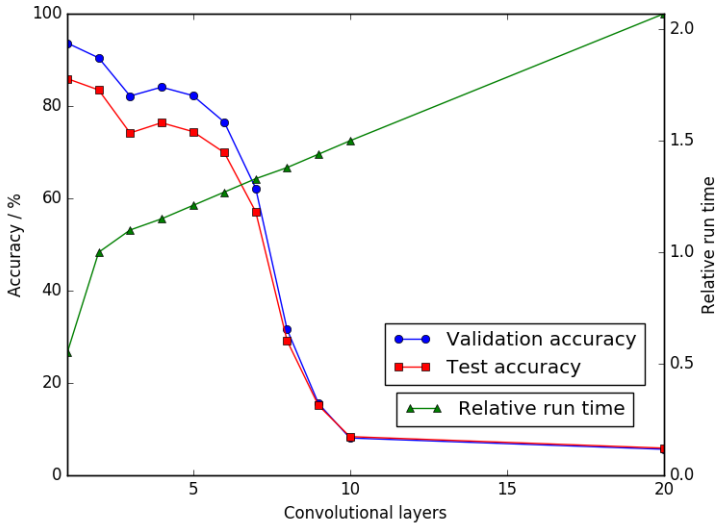
10 epochs, 2 pooling layers

Conv. layers	Validation acc.	Test acc.	Training time
2	**90.44 %**	**83.48 %**	**3 m 55 s**
3	92.83 %	85.14 %	4 m 23 s
4	93.22 %	86.21 %	4 m 51 s
5	93.81 %	87.13 %	5 m 20 s
6	93.07 %	86.74 %	5 m 48 s
7	92.18 %	85.57 %	6 m 18 s
8	92.34 %	85.94 %	6 m 45 s
9	84.65 %	79.03 %	7 m 17 s
10	75.88 %	70.45 %	7 m 45 s
20	5.76 %	5.91 %	12 m 30 s

500 epochs, 2 pooling layers

Conv. layers	Validation acc.	Test acc.	Training time
2	99.77 %	96.62 %	3 h 13 m 29 s
3	99.69 %	96.11 %	3 h 39 m 37 s
5	99.51 %	95.15 %	4 h 29 m 28 s
10	99.24 %	94.20 %	6 h 26 m 19 s

Figure 4.24: Impact of increasing the number of convolutional layers in the baseline architecture, by stacking them after the second pooling layer. No more pooling layers are added, to keep the minimum spatial size to 8×8. The top table shows the results when running each fold in the 10-fold cross validation for ten epochs, and is also visualised in the graph, while the bottom table contains the results when increasing the number of epochs to 500. The top table also contains the baseline architecture results, which is shown in bold.



10 epochs, 3 pooling layers

Conv. layers	Validation acc.	Test acc.	Training time
2	**90.44 %**	**83.48 %**	**3 m 55 s**
3	82.17 %	74.16 %	4 m 18 s
4	84.09 %	76.38 %	4 m 30 s
5	82.26 %	74.46 %	4 m 44 s
6	76.51 %	69.91 %	4 m 58 s
7	62.17 %	57.10 %	5 m 13 s
8	31.59 %	29.16 %	5 m 24 s
9	15.53 %	15.12 %	5 m 38 s
10	8.08 %	8.38 %	5 m 53 s
20	5.64 %	5.89 %	8 m 6 s

500 epochs, 3 pooling layers

Conv. layers	Validation acc.	Test acc.	Training time
2	99.77 %	96.62 %	3 h 13 m 29 s
3	99.69 %	96.64 %	3 h 37 m 7 s
5	99.24 %	93.91 %	3 h 49 m 41 s
10	98.96 %	93.17 %	4 h 54 m 24 s

Figure 4.25: Test results when increasing the depth of the network, baseline architecture being two convolutional layers. A pooling layer is added after each of the first three convolutional layers 8there are only two pooling layers in the instance with just two convolutional layers), but not after the following layers, to keep minimum size of images to 4×4. The top table contains the results when running the network for ten epochs in each fold of the 10-fold cross validation, and is also visualised in the graph above. The bottom table contains the test results from increasing the number of epochs to 500 for each fold.

4.2.2.7 Linear Rectifier

Table 4.7 shows the results of applying a non-zero gradient for negative input according to equation (2.1) in section 2.1.3.1. This is applied to the rectifiers after the two convolutional layers and fully connected layers. The fold accuracy varied with just a couple of percentage units.

Table 4.7: The results when applying a non-zero gradient for negative input to the rectifiers after the convolutional layers and the fully connected layers at the end. For details on rectifiers, see equation (2.1) in section 2.1.3.1. The top table contains the results when running each fold in the 10-fold cross validation for ten epochs, the bottom when increasing the number of epochs to 500.

10 epochs

Gradient	Validation acc.	Test acc.	Training time
0.00	**90.44 %**	**83.48 %**	**3 m 55 s**
0.01	90.11 %	82.84 %	3 m 57 s
0.33	93.72 %	87.16 %	3 m 55 s

500 epochs

Gradient	Validation acc.	Test acc.	Training time
0.00	99.77 %	96.62 %	3 h 13 m 29 s
0.33	99.69 %	96.59 %	3 h 16 m 53 s

Introducing a gradient for negative input to the rectifiers in the convolutional and fully-connected layers appears to have an impact, but only when using a high gradient of 0.33 and the gain is only a few percentage units, as can be seen in table 4.7. When the number of epochs is increased to 500 during training, the impact of the rectifier is almost non-existent, meaning it only seems to make the network converge faster, not achieve a better results when converged. Using a gradient requires very few extra calculations and training time is not affected.

4.2.2.8 Pooling Layer

In table 4.8 the results of alternating the filter size of the pooling layers can be seen. The stride is the same as the filter size and no zero padding is added. The baseline architecture is size 2×2. Also note that size 1×1 is the same as no pooling, i.e. the layer outputs the input unchanged. Fold accuracy varied with just a few percentage units for the two smaller filter sizes and a little more for filter size 4×4, with 5 %.

Pooling layers are used to downsample the feature maps, which decreases training time and potentially avoids overfitting, table 4.8 shows the effect of alternating the filter size of the pooling layers in the baseline architecture. This is a destructive operation, which can clearly be seen when using 4×4 filter size, the results are significantly worse than when using 2×2, as in the baseline architecture. Not surprisingly, the validation accuracy is really high after just ten epochs when using a filter size of 1×1, which is the same as not applying a pooling layer. No information

Table 4.8: Results of using different filter sizes for the max pool layers, stride is the same as the size and no zero padding is added. Size 1×1 will have the same effect as no pooling, i.e. outputs the image unchanged, and the baseline architecture is 2×2. The top table contains the results when training for ten epochs for each fold in the 10-fold cross validation, and the bottom when increasing the number of epochs to 500.

10 epochs

Size	Validation acc.	Test acc.	Training time
1×1	98.16 %	91.56 %	11 m 59 s
2×2	**90.44 %**	**83.48 %**	**3 m 55 s**
4×4	52.71 %	48.16 %	2 m 12 s

500 epochs

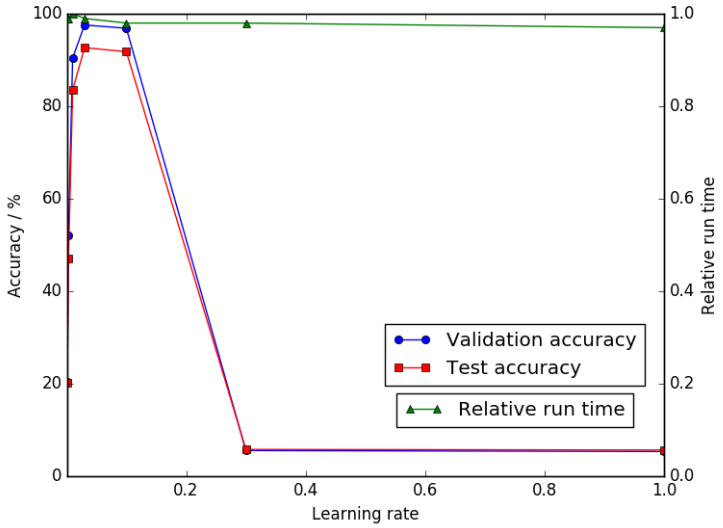
Size	Validation acc.	Test acc.	Training time
1×1	99.55 %	95.18 %	10 h 1min 38 s
2×2	99.77 %	96.62 %	3 h 13 m 29 s

is lost, but training time increases since the image size is kept the same, thus leading to more computations throughout the network. When the network is allowed to converge better while running for 500 epochs the accuracy without pooling layers decreases. This is expected as pooling layers should help prevent overfitting.

4.2.2.9 Learning Rate

Figure 4.26 displays the impact alternating the learning rate had. The fold accuracy varied the least for a learning rate of 0.03. The fold accuracy for a learning rate of 0.1 varied between 88.46 % and 98.86 %. Loss functions for learning rate 0.03 and 1.0 can be seen in figure 4.27, which is the categorical cross-entropy, defined in section 2.1.3.1 as equation (2.3).

Higher learning rate can increase the speed of learning. The learning rate determines the magnitude of the updates the network should make after each batch, so a larger learning rate takes bigger steps in the "right direction". However, using a learning rate that is too large can cause the loss function to never converge to a minimum, which is probably the case for learning rates 0.3 and 1.0 in the table in figure 4.26. It takes too great steps in the right direction and overshoots the optimal values. In figure 4.27 it can be observed how the loss function of a high learning rate never changes while a lower learning rate gradually converges. For this network, a learning rate of 0.03 seems to lead to quick convergence and significantly improves the results compared to the baseline architecture after just ten epochs. The holds true for an increased number of epochs, the network converges to a higher accuracy with the higher learning rate. The time to train is constant since no extra calculations are made.



10 epochs

Learning rate	Validation acc.	Test acc.	Training time
0.001	20.48 %	20.24 %	3 m 53 s
0.003	51.98 %	47.13 %	3 m 53 s
0.01	**90.44 %**	**83.48 %**	**3 m 55 s**
0.03	97.58 %	92.69 %	3 m 53 s
0.1	96.87 %	91.79 %	3 m 50 s
0.3	5.55 %	5.79 %	3 m 50 s
1.0	5.34 %	5.67 %	3 m 48 s

500 epochs

Learning rate	Validation acc.	Test acc.	Training time
0.01	99.77 %	96.62 %	3 h 13 m 29 s
0.03	99.83 %	97.11 %	3 h 15 m 54 s

Figure 4.26: Test results for various learning rates, using 10-fold cross validation. The top table contains the results when running each fold in the 10-fold cross validation for ten epochs, and contains the baseline architecture in bold. It is also visualised in the graph above, where training time is shown as relative to the baseline architecture. The bottom table contains the results when increasing the number of epochs to 500.

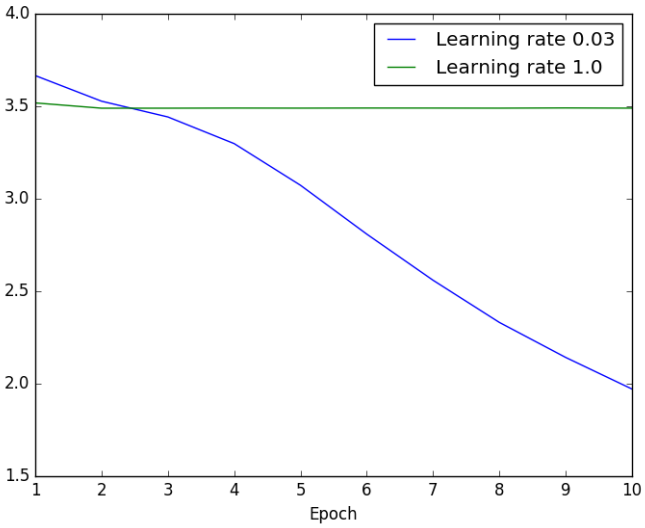
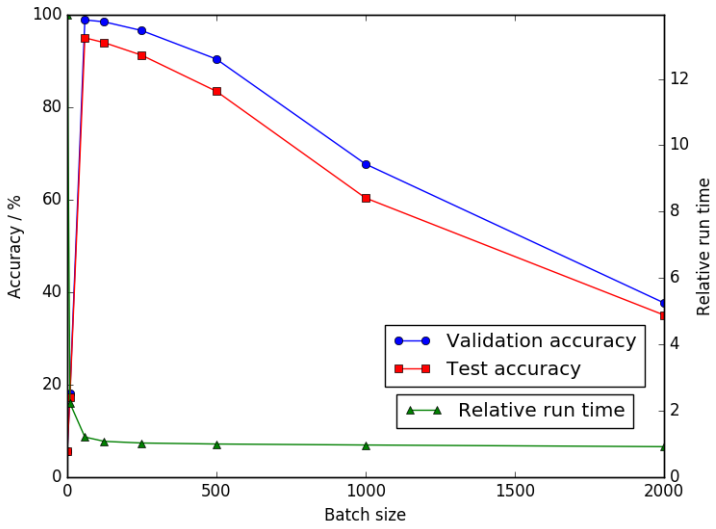


Figure 4.27: Loss functions for learning rate 0.03 and 1.0 using the baseline architecture. Categorical cross-entropy was implemented as loss function, details can be found in section 2.1.3.1.

4.2.2.10 Batch Size

In figure 4.28, the results of varying the batch size used for training can be found. The fold accuracy varied the least for a batch size of 60, 125, and 250, with just 1-2 percentage units.



10 epochs

Size	Validation acc.	Test acc.	Training time
1	5.55 %	5.65 %	54 m 34 s
10	18.14 %	17.33 %	8 m 44 s
60	98.90 %	95.02 %	4 m 44 s
125	98.48 %	93.98 %	4 m 14
250	96.60 %	91.27 %	4 m 2 s
500	**90.44 %**	**83.48 %**	**3 m 55 s**
1000	67.73 %	60.39 %	3 m 48 s
2000	37.70 %	35.07 %	3 m 36 s

500 epochs

Size	Validation acc.	Test acc.	Training time
60	97.27 %	91.97 %	3 h 58 m 22 s
500	99.77 %	96.62 %	3 h 13 m 29 s

Figure 4.28: Accuracy and relative run time for various batch sizes, the top table contains the results when training for ten epochs, which are also visualised in the graph above, and the bottom when running for 500 epochs. Baseline architecture is shown in bold in the top table.

The table in figure 4.28 contains the quite interesting results of alternating the batch size, a smaller batch size appears to generate better results when training for ten epochs. When it gets too small however, the accuracy deteriorates drastically. For large batches, the program is constructed in such a way that for the last batch, if the images do not make up a full batch the remaining images are discarded, therefore

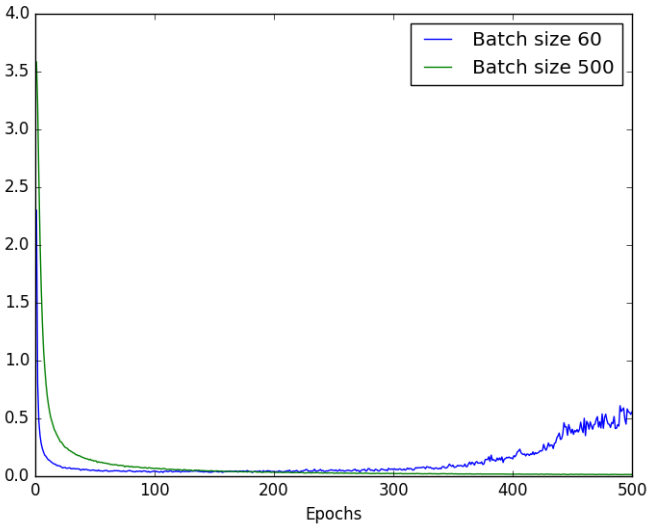


Figure 4.29: Loss functions for batch sizes 60 and 500 when run for 500 epochs. Batch size 60 shows clear evidence of overfitting.

losing some information (images), leading to worse performance. Another reason for smaller batch size generating better results after the same number of epochs is assumed to be due to the fact that the weights are updated more often during each epoch. If the batch size is too small the updates are more random since they will only adjust for very few images that might not be representative of the whole dataset, and the adjustments might therefore not be necessarily statistically accurate. Using smaller batch sizes means the weights are updated more often, requiring additional computations. This is assumed to be the reason why the training time is higher with smaller batch size.

When increasing the number of epochs to 500, the larger batch size generates significantly better results. The accuracy when using the smaller batch size has actually decreased, particularly the test accuracy. This can in part be because it only updates the network after using 60 images, meaning it is unlikely that all classes are represented in each update. But when looking at figure 4.29 where the loss function for 60 epochs is compared with the loss function for 500 epochs, it can be seen that the loss function first converges before starting to diverge. This is a clear indication that overfitting has occurred where the network has become over-specialised on the training images, and no longer works well on other images it has not been trained on. The reason it happens for batch size 60 and not 500 is since the updates on the network occurs more often, this makes the network converge faster, but also causes it to diverge earlier.

4.2.2.11 Input Image Size

Table 4.9 displays the impact of alternating the size of the input image. The larger image sizes showed smaller variance for the fold accuracies when run for ten epochs, with less than one percentage unit for image size of 64×64.

Table 4.9: Results of varying image input size, the top table when training for ten epochs and the bottom when training for 500 epochs. baseline architecture is shown in bold in the top table.

10 epochs

Size	Validation acc.	Test acc.	Training time
8	32.21 %	29.06 %	28 s
16	62.34 %	56.17 %	1 m 3
32	**90.44 %**	**83.48 %**	**3 m 55 s**
64	97.93 %	91.81 %	15 m 19 s
128	Out of memory		

500 epochs

Size	Validation acc.	Test acc.	Training time
32	99.77 %	96.62 %	3 h 13 m 29 s
64	99.74 %	96.56 %	12 h 55 m 6 s

It can be observed in table 4.9 that image size greatly affects both training time and accuracy. When downscaling images, information is lost with the lowered resolution, it is therefore not strange that the performance declines. However, with too large images, mainly memory but also training time become issues to consider when building a CNN. Enlarging the images also means that their properties change, which can alter the way the convolutional layers view them. For example edges become more diffuse, meaning filters of small size could have a harder time detecting them. The largest image size that could be tested without running into memory issues was 64×64 pixels. This is reasonably close to the median and average sizes of 43×43 and 56×56 pixels respectively. Distortion by enlarging the images have thus been kept quite low and the results indicate that the performance increases due to the extra information available with larger image sizes.

When training for 500 epochs, the image size seems to not matter greatly. The smaller image size even generated slightly better results, and this combined with the significantly increased training time makes larger image sizes very disadvantageous to use in comparison with a smaller size.

4.2.3 Dataset Analysis

An observation that can be seen in virtually all of the test cases is that the validation accuracy is notably higher than the test accuracy. The reason behind this is most likely because the dataset actually contains many images of the same signs from different distances and angles, but with the same weather conditions and surrounding noise, such as background and objects partly blocking view of the signs. This means the images can be very similar, especially with regards to lighting and background noise. When the dataset is loaded, all the images are randomly shuffled to attain a mix of classes in the training and validation sets. Because of this, the validation data is likely to contain images correlated to images used for training, leading to higher accuracy since the network has been trained on very similar images. This is

not the case with the testing data, which stems from a completely separate set of images of other traffic signs, and thus the accuracy is considerably lower but can be viewed as the ability of the network to generalise to previously unseen datasets.

4.3 Discussion

The best results achieved, 98.81 %, were generated by creating an ensemble out of the best performing network, which was a slightly larger network than the baseline architecture, in addition to making use of longer training runs and a slightly larger learning rate than in the initial design. The performance was just below that of humans on the same dataset, 98.84 %, which shows how powerful even a relatively simple CNN can be.

Of course it is always possible (in theory at least) to conduct further testing to try to improve the results, and this is no exception. The results for class 28 for example, show that CNNs can detect features that are not important to the subject at hand. This can be alleviated by some form of pre-processing to remove unwanted features. It would also be possible to add handcrafted features as input in addition to the images. All hyperparameters could be more finely tuned or combined in other ways to see if any small changes would yield better results. Using a different, perhaps larger, baseline architecture might also generate different results that may (or may not) be worth looking into. If time permits, running training for more epochs is also desirable since this should increase the accuracy at least some, as long as it does not introduce overfitting. By analysing the loss function it would be possible to see how far the network can be pushed, i.e. for how many epochs it can run, before overfitting becomes an issue.

Due to the nature of CNNs it is often difficult to understand why they get the results they do, and by extension how they best can be improved upon. Often it is impossible to know what changes need to be made, and the only way to find out is to test extensively, which quickly becomes time consuming. A good plan to structure training and testing, as well as being aware of if the plan at some point should be deviated from, is highly recommended. A decent GPU definitely helps as well.

5

Detection of Alzheimer's Disease

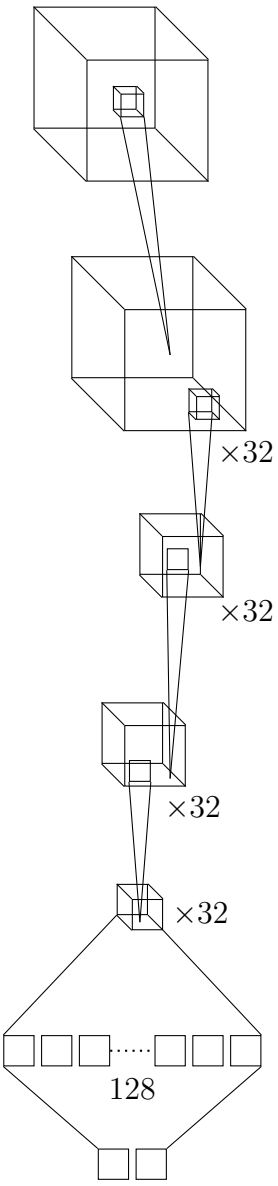
This section aims to describe how the detection of Alzheimer's disease from MRI images was conducted, beginning with a description of the investigated methods in section 5.1, followed by the results and performance evaluation in section 5.2 and a general discussion of the results in section 5.3.

5.1 Methods Investigated in this Thesis

Diagnosing Alzheimer's disease is no trivial task. This section aims to describe how a deep learning classification system for attempting to do just that was set up, beginning with how training and testing was conducted, followed by an explanation of the datasets used, and lastly a short description of the implementation of the program itself.

5.1.1 Training, Validation, and Testing

The dataset used stems from the Alzheimer's Disease Neuroimaging Initiative (ADNI) as i described more indepth in section 5.1.2. Testing on the ADNI dataset was not as extensively conducted as for the GTSRB dataset, described in section 4. The network structure used was very similar to the baseline architecture described in section 4.1.1 and can be seen in figure 5.1. The biggest difference between the two structures is that the input images are three dimensional MRI images instead of regular RGB images, leading to three dimensional computations. The input consists of one channel (gray scale) MRI images that have been rescaled to size $64 \times 64 \times 64$ voxels (3D pixel). It was sent into the network in batches of only two at a time due to memory constrainst preventing larger batches from being used. First comes one 3D convolutional layer with 32 $5 \times 5 \times 5$ filters, zero padding is two on all sides of the image (cube) to maintain the size after convolution. A stride of one is used in each dimension. The convolutional layer is then followed by a linear rectifier and a max pooling layer, with spatial filter size $2 \times 2 \times 2$, which downsamples each dimension to half its size, i.e. the output is now 32 feature maps of spatial size $32 \times 32 \times 32$. Then another 3D convolutional layer, a linear rectifier, and one more pooling layer with the same settings follows. The feature maps are now of spatial size $16 \times 16 \times 16$. Finally, the network ends with two fully connected layers, the first with 128 units and the second with two output units (the class probabilities), with a linear rectifier between them. During training, dropout rate of 0.5 is applied on both layers. The final output layer applies the softmax function to calculate the class probabilities.



Input: MRI image, size $64 \times 64 \times 64$

3D Convolution: 32 filters, spatial size $5 \times 5 \times 5$, zero padding 2, stride 1
ReLU: $f = max(0, x)$
Outputs 32 feature maps of spatial size $64 \times 64 \times 64$

MaxPool: spatial size $2 \times 2 \times 2$, no padding, stride 2
Outputs 32 feature maps of spatial size $32 \times 32 \times 32$

3D Convolution: 32 filters, spatial size $5 \times 5 \times 5$, zero padding 2, stride 1
ReLU: $f = max(0, x)$
Outputs 32 feature maps of spatial size $32 \times 32 \times 32$

MaxPool: spatial size $2 \times 2 \times 2$, no padding, stride 2
Outputs 32 feature maps of spatial size $16 \times 16 \times 16$

FC: 0.5 dropout rate, ReLU
Outputs 128 units

FC: 0.5 dropout rate, softmax
Outputs 2 units (class probabilities)

Figure 5.1: Baseline architecture used for training and testing on the ADNI dataset.

All tests were run for 50 epochs, using one training set and one validation set, no cross-validation was applied. Testing was done using both the original images and slightly cropped images, with the aim of focusing more on the brain and minimizing the surrounding skull and eyes etc. The cropping was done non-uniformly, since the images themselves are not uniform. The zero rule, **ZeroR**, was also applied for comparison. ZeroR classifies everything as the majority class of the training example and is a simple classification method often used as a benchmark for classification. It has higher accuracy on a binary classification problem than simply tossing a coin (which should have 50 % accuracy, if the coin is fair), how high depends on the class distributions, but the chosen classification method should *at least* generate as high

accuracy as that of ZeroR, otherwise the chosen method can be deemed useless.

5.1.2 Dataset

The dataset used in this thesis was obtained form the Alzheimer's Disease Neuroimaging Initiative (ADNI) database (adni.loni.usc.edu). ADNI is a global project that makes reliable clinical data available to researchers of Alzheimer's disease. It was launched in 2003 as a public-private partnership, led by Principal Investigator Michael W. Weiner, MD. ADNI aims to test if it is possible to combine serial magnetic resonance imaging (MRI), positron emission tomography (PET), other biological markers, and clinical and neuro-psychological assessment to examine and measure the progression of mild cognitive impairment (MCI) and early Alzheimer's disease. The data itself is collected, validated, and utilized by ADNI researchers[8].

Two different types of MRI images are used in this thesis to train the network separately. The first type is regular MRI with T1 weighing. They provide a full picture of the body clearly showing both bone and softer tissue. The second type is Diffusion Tensor Imaging (DTI). These images shows the diffusion of water in tissue and thus highlight the brain more, since ample diffusion takes place there.

Both of the image types are split into one training and one validation set. The sets are split in such a way that no patient exists both in the training and the validation set, in order to avoid correlation between the sets. Since several patients have images taken at different visits over the years, images of the same patient can exist within either the training or validation sets, but never in both. The DTI dataset also has multiple images of the same patient taken at the same visit and therefore the full dataset was split into both a larger and a smaller dataset. In the smaller dataset only one image from each visit was used and in the larger all images from the visits were used.

In total 826 regular MRI images were used, 77 % (634) of them for training and the remaining 23 % (189) for testing. For the DTI images 378 images made up the smaller dataset, 75 % (282) of them were used for training and 25 % (96) for testing, and $10,886$ made up the larger dataset, which was split up into 77 % (8410) for training and 23 % (2476) for testing. The distribution of AD and normal images in the different sets used can be viewed in figure 5.2

Figure 5.3 shows the center slices of two example images from the regular MRI dataset, one from a patient diagnosed with Alzheimer's disease and one from a healthy person. As mentioned, the images were also cropped, an example of which can be seen in figure 5.4 where the examples images from figure 5.3 have been cropped to remove some of what is not a part of the brain itself.

Figure 5.5 displays DTI images of a healthy brain and one afflicted by AD. As can easily be observed the brain is much better highlighted when compared to the images in figure 5.3

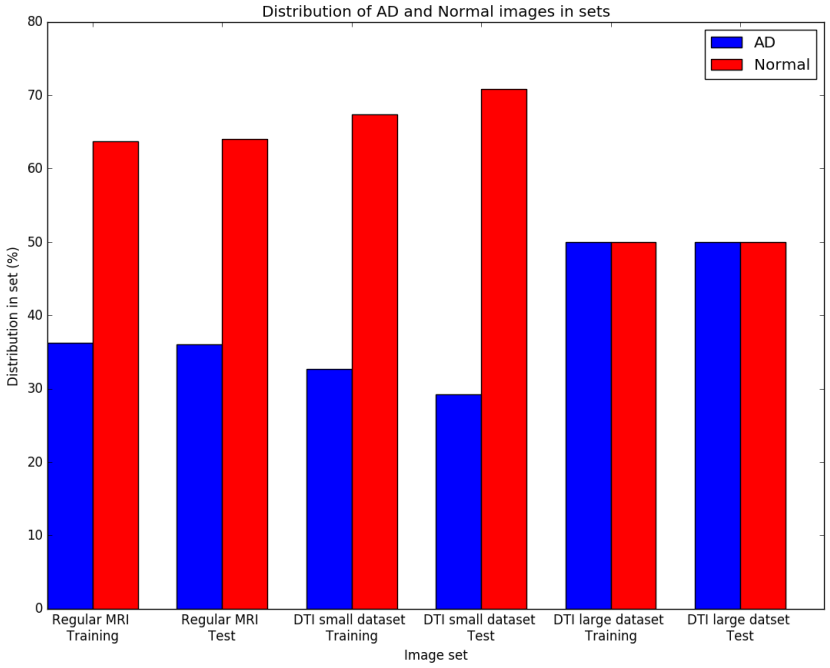
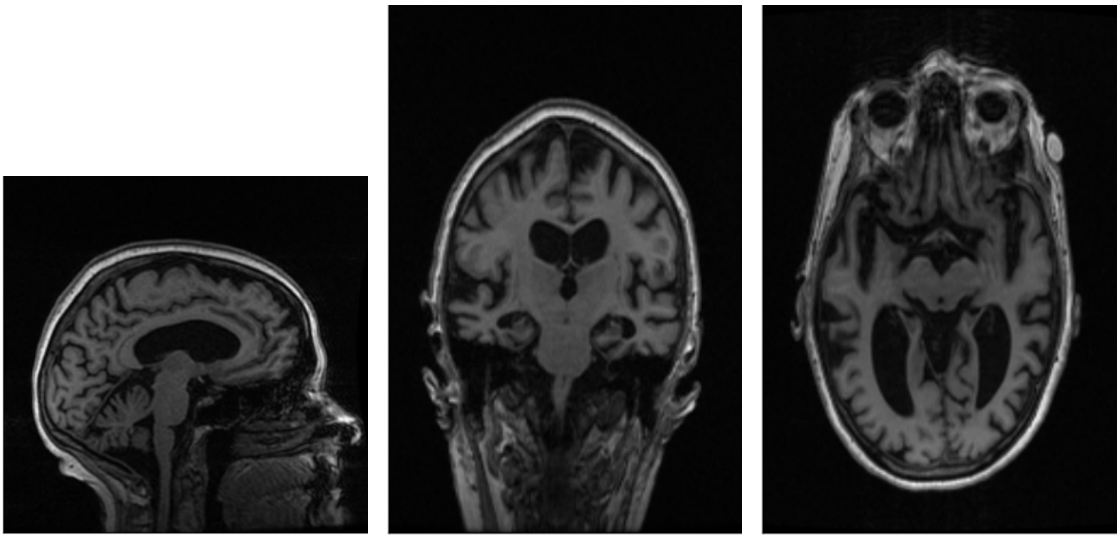
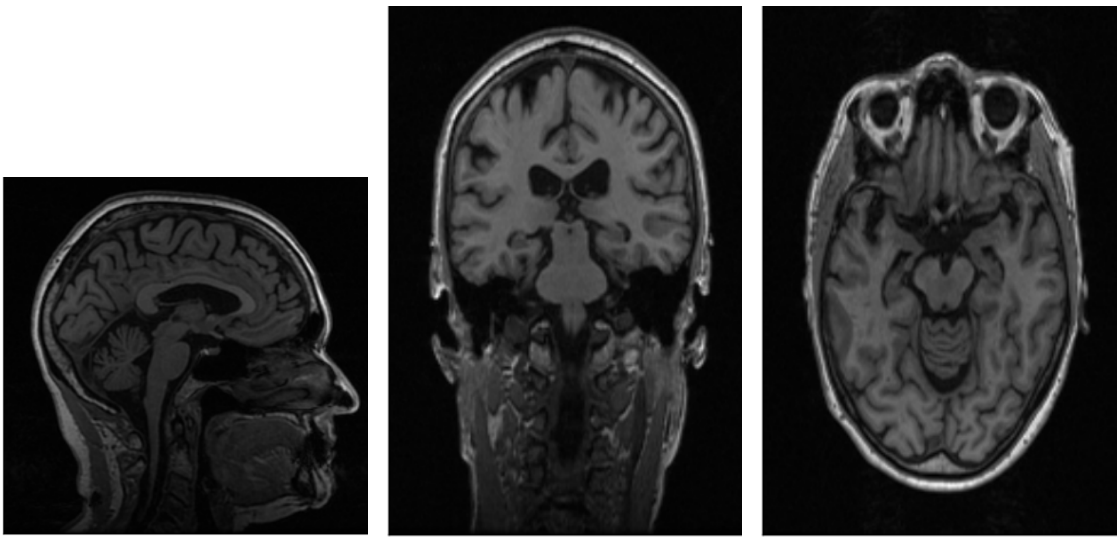


Figure 5.2: Distribution of images showing brains with and without AD in the used datasets. Distribution is shown in percentage. The total number of MRI images used are 826, the small DTI dataset consists of 378 images, and the large of 10,886 images.

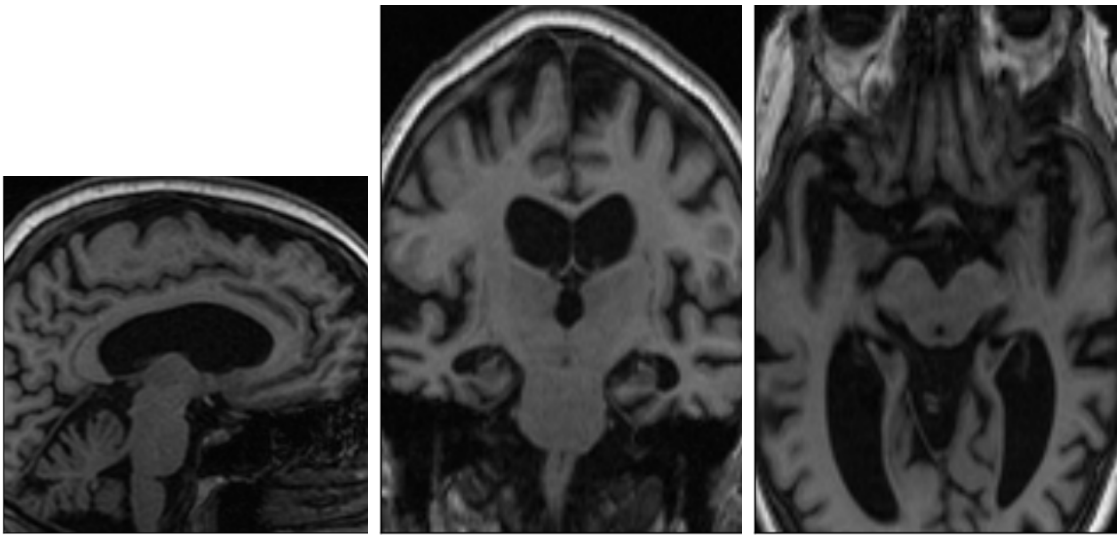


(d) MRI images from a patient with Alzheimer's disease.

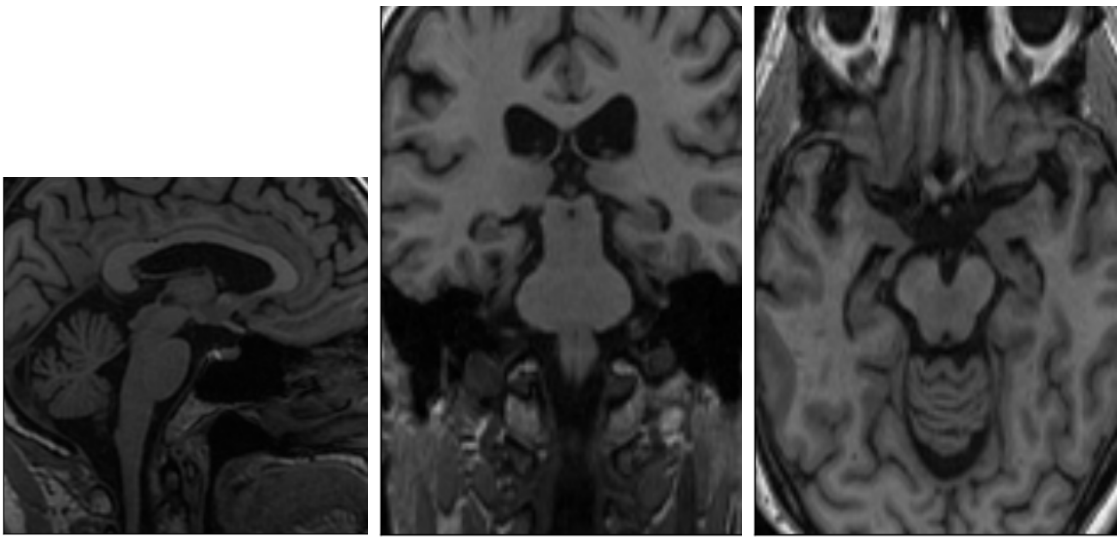


(h) MRI images from a healthy person.

Figure 5.3: Example MRI images from the ADNI dataset, one with Alzheimer's disease (top) and one healthy person (bottom)[8].

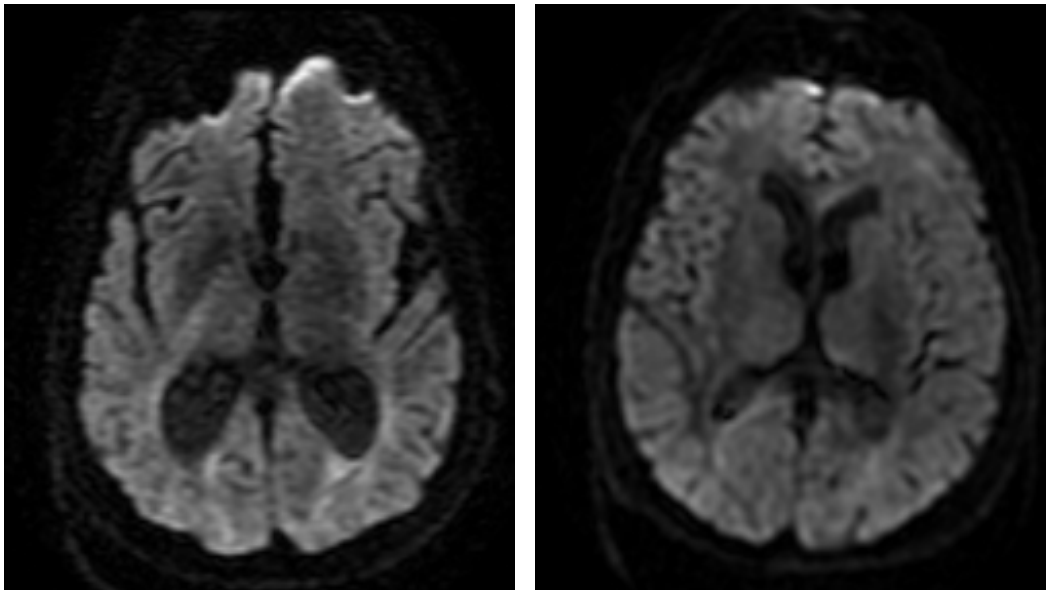


(d) MRI images from a patient with Alzheimer's disease.



(h) MRI images from a healthy person.

Figure 5.4: Example of how the images were cropped to enable the brain itself to take up a larger part of the image.



(a) DTI image of brain with AD **(b)** DTI image of healthy brain

Figure 5.5: Examples of DTI images from the ADNI dataset[8].

5.1.3 Implementation

For the implementation of this system, the code used for the traffic sign recognition part was modified to be able to load the dataset using the Python library NiBabel. The network itself was also modified to be able to handle three dimensional images, as well as designed to classify the new images.

For this implementation, the Keras library was used instead of Lasagne. Keras allows for fast and easy prototyping, and like Lasagne uses Theano, which made it appropriate to use for this problem and greatly simplified the implementation process.

5.2 Results and Performance Evaluation

The results from testing described in section 5.1 are presented in tables 5.1 and 5.2, when using the regular MRI images and DTI images respectively. The first column is the name of the test, the second the accuracy on the dataset used for training, and the third column the test accuracy on the completely separate test dataset, both after 50 epochs of training. The network structure used is described in figure 5.1 in section 5.1.1. Cropping of the regular MRI images and the small dataset of DTI images was conducted non-uniformly, i.e. differing amounts for each dimension of the images, to better fit the brains in the images. The large dataset of DTI images was not cropped. In the tables can also the benchmark results obtained from applying the zero rule, ZeroR, be found, which is also described in section 5.1.1.

Table 5.1: Results when using regular MRI images from the ADNI dataset, both when using the original images, slightly cropped images, and compared to the benchmark given by the zero rule, as explained in section 5.1.1, where also a detailed description of the network used can be found. Training accuracy is the accuracy on the dataset used for training, while test accuracy is the accuracy on a completely separate dataset. All training was run for 50 epochs.

50 epochs

Architecture	Train acc.	Test acc.	Training Time
MRI	97.79 %	58.20 %	1 h 28 m
MRI Cropped	98.90 %	58.73 %	1 h 28 m
ZeroR	62.73 %	64.02 %	–

Table 5.2: Results when using the DTI images from the ADNI dataset, one small dataset containing only one image from each patient and one larger containing multiple images from the same patient. The images in the smaller dataset were also cropped for one test case. Comparison with the benchmark accuracy obtained from applying the zero rule can also be seen, which is described in section 5.1.1, where also a detailed description of the network structure can be found. Training accuracy is the accuracy on the dataset used for training, while test accuracy is the accuracy on a completely separate dataset. All training was run for 50 epochs.

50 epochs

Architecture	Train acc.	Test acc.	Training Time
DTI small	92.20 %	59.38 %	40 m
DTI small cropped	91.13 %	53.12 %	43 m
DTI small ZeroR	67.37 %	70.83%	–
DTI large	99.98 %	65.19 %	19 h 36 m
DTI large ZeroR	50.01 %	50.00 %	–

5.3 Discussion

Table 5.1 and table 5.2 both show that detection of AD is a much more complex task, compared to classifying traffic signs. ZeroR actually outperforms the trained network in all cases but one and overfitting of the network is obvious, which can be seen when comparing the accuracies on the training sets and test sets. One reason behind the poor results can be the network structure, due to limitations in hardware a larger network could not be tested. Another probable reason is the small number of images in the datasets used, CNNs are known for needing big quantities of images to be able to train properly. It can be observed that in the case when many images were available for training, as was only the case for the large DTI dataset, the network accuracy beat the ZeroR accuracy by 15.19 percentage units, showing clear signs that it was able to learn at least some differences between a healthy brain and one with AD. Even though many of the images in this dataset were very similar, as several scans of the same patient were recorded during the visits, the extra data appears to have helped train the network better. However, when separating the datasets into training and testing, it was made sure that no patients would appear on both, so as not to create any correlation between the datasets.

Other things that can be seen is that trimming the images to show primarily the brain actually has either no or even a detrimental effect on the results. This would suggest that having a large blank space in the images does not affect the results very much.

No larger differences are observed between using regular MRI or DTI images except for the large DTI dataset, the main cause of this is probably the number of images. But since no more regular MRI images were available it could not be tested to see if the results would have been comparable. Intuitively, the DTI images would appear to be better suited for the task, since they highlight the brain better and thus provide a clear area of interest for the network. However, CNNs are not easily

analyzed and the network could be able to identify features not clearly visible to the human eye.

Using Keras instead of Lasagne revealed some limitations with the library. The system quickly ran out of memory when trying out larger network designs, which limited the size of networks that could be tested. The larger size of the input data to the network also contributed to this. Since input size of the images were both higher resolution and in 3D, compared to the traffic signs, it was very time consuming to train the network. Even the small DTI dataset consisting of just under 400 images took around 40 minutes to train on the relatively simple network, and this was only for a single run over data, not 10 as was done for the traffic signs. Because of this, 10-fold cross-validation was not used as it was deemed too resource intensive and did not provide enough of an advantage to justify the extra costs in run time. The large DTI dataset would have taken over a week to train using 10-fold cross validation, which was not considered justified. Note that only the training time for the network is shown in the tables, not including loading and reshaping the images. It took considerably longer time to transform and load the larger 3D images into memory when compared to the traffic sign dataset. In the case of the large DTI dataset it took close to two hours for this process, the traffic sign dataset took under a minute. This can of course be alleviated by storing the matrices containing the transformed image data as binary files ready to be loaded into python quickly. This would have been almost a necessity if further testing on the subject had been done.

The explanation for ZeroR receiving a higher test accuracy than training accuracy, which is generally not possible with machine learning algorithms, is because it only takes which class is the majority class in the training dataset into consideration, and then classifies everything in the test dataset as this class. This means the accuracy is only dependant on the distribution of the majority class in the two sets, which usually differs and in this case the majority class of the training dataset makes up a larger part of the test dataset. The class distributions can be seen in figure 5.2 in section 5.1.2.

To summarise, the results are not excellent but lays the ground for further research. The tests were performed mainly to see whether a simple network has the potential to solve this type of classification problem, not with the intention of creating a perfectly functioning system. The indications are that larger datasets, and probably deeper networks, would be the best candidates to test in future work. However, this would also require a much more powerful computer than used in this study. Additionally, it would also be interesting to investigate detection of early signs of Alzheimer's, which are more subtle and difficult to distinguish, but could prove extremely beneficial for the research community if successful.

6
Ethical Aspects and Sustainability

In this section ethical aspects and sustainability is discussed. Present and future issues that can arise concerning machine learning and its applications are brought to light, along with problems it may solve.

6.1 Machine Learning and Artificial Intelligence

The digitalised society has led to an enormous amount of data becoming available for analysis. Data is collected all the time when we use our phones and computers, both in ways we are aware of and in ways we may not be. Companies like Facebook and Google use this today to map our interests and provide us with services that tailors to these interests. Though this clearly has advantages it also causes concerns for how it affects our integrity. How much information should be collected, who can use it and in which ways? Do the benefits of the developed system always justify the costs, e.g. on personal integrity? These are important questions that should always be considered when working with personal data.

6.2 Traffic Sign Recognition and its Areas of Use

There are several aspects to consider regarding ethics and sustainability of traffic sign recognition and its potential uses. What kind of data is collected and how is it used? If images are collected by the vehicles, not only traffic signs will be present but also people, registration plates, etc. How this information is stored and who has access to it is important to be aware of for the integrity of those present along the road.

Another important ethical aspect is who is liable if or when something goes wrong, e.g. the speed limit is displayed inaccurately in an assistance driving program, can the developers behind the system be held liable for fines received while driving over the speed limit? More serious or even fatal situations can arise when autonomous driving systems are being used, can the creators behind the vehicle be held liable then? Or is it the sole responsibility of the person behind the wheel, even though that person was not in actual control of the vehicle? These are important questions to consider before putting these kinds of systems to use, both for the companies developing them, legislators, and end users of such systems and products.

From a sustainability viewpoint, perhaps autonomous driving can be an improvement on the environment, if it can better plan its driving than humans. Trans-

port can also be done cheaper and any time of day, any time of the week. How this will affect the job market however, is a whole other matter to consider.

6.3 Alzheimer's Disease Detection and Medical Applications

The average age of the population in the industrialized world is ever increasing. This will lead to more people requiring medical attention, which will strain the health care of today. A change to make health care more efficient is required and automated systems that can either do complete diagnosis, or just assist in doing so would be tremendously beneficial. It is also possible that computers in many regards could be more reliable since they do not tire and never forget. They can make use of huge amounts of data in their analysis and would be able to perform better with more experience.

Apart from performing the analysis done today, new types of diagnosis might become available to detect diseases at an early stage allowing, for faster and better treatment and preventing complications that would arise if the disease is allowed to progress. This would mean better quality of life for the patients and a lower cost for the treatments. Since machine learning algorithms often can find correlations humans cannot this is not outside the realm of possibility and would be vary valuable.

There is always the question of responsibility, which is even more present in the health care sector where the lives of people are directly affected. If a person is harmed due to a computer error, who is to be held liable? Even if a computer just assists a doctor, what if a diagnosis is made based on erroneous results, can the doctor be held responsible for trusting the system? Or the developers behind the system? These questions are very central and have no simple answers. Since deep learning algorithms can be difficult to analyse there is the added factor that one might not know exactly how the computer acquires its results and therefore it might be hard to prevent errors from happening again. There is much work to be done here in order to protect both the patients, but also the individual doctors using the systems.

7
Conclusions

To sum up the findings of this thesis, relatively simple CNNs perform well on traffic sign classification, but not as well on Alzheimer's disease detection. The differences in performance could be attributed both to the amount of data available, and also to that AD is a more complex classification problem due to the visual differences being more subtle. To further study this problem, a deeper network is suggested to be used. This could potentially increase the performance, however, this would also require significantly more computational power. Additionally, a larger dataset would likely increase performance and reduce overfitting. It is well known that for deep learning problems, a large dataset is needed to accurately train the weights of the algorithm used.

Alternating the hyperparameters can be beneficial to finding an optimal network structure, but also time-consuming and should be done with care. One must also consider that the network finds all types of similarities between the training images, not just the ones relevant to the problem at hand. Testing showed that performance gains were relatively minor when comparing larger networks to smaller ones on the traffic sign classification problem. There needs to be a balance between computational cost, i.e. run time and available memory, and the network performance.

Even though it is difficult to analyse what really happens between the input nodes and output nodes of a CNN, their use is of great advantage for image analysis since they scale well when compared to other available machine learning algorithms. There have been great advancements in the field of deep learning in recent years, the reason being both the amount of data readily available and increase in computer power. It is a technology already used by major corporations today and will likely remain so for the foreseeable future.

Bibliography

[1] Embedded Vision Alliance. Using convolutional neural networks for image recognition. http://www.embedded-vision.com/platinum-members/cadence/embedded-vision-training/documents/pages/neuralnetworksimagerecognition, 2017. Accessed: 2017-01-17.

[2] Cambridge Coding Academy. Deep learning for complete beginners: Using convolutional nets to recognise images. http://online.cambridgecoding.com/notebooks/cca_admin/convolutional-neural-networks-with-keras, 2017. Accessed: 2017-01-17.

[3] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[5] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[6] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[7] Google Inc. Google self-driving car project, monthly report, march 2016. https://static.googleusercontent.com/media/www.google.com/en//selfdrivingcar/files/reports/report-0316.pdf, March 2016. Accessed: 2016-11-24.

[8] Alzheimer's Disease Neuroimaging Initiative. Adni. http://adni.loni.usc.edu/, 2017. Accessed: 2017-01-03.

[9] Alan Turing. I.—computing machinery and intelligence. *Mind*, LIX(236):433–460, 1950.

[10] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[11] Ben Kröse, Patrick van der Smagt, and Patrick Smagt. An introduction to neural networks, 1993.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.

[13] Fei-Fei Li, Andrej Karpathy, and Justin Johnson. Lecture notes in cs231n convolutional neural networks for visual recognition: Lecture 7 convolutional neu-

ral networks. http://cs231n.stanford.edu/slides/winter1516_lecture7.pdf, January 2016. Accessed: 2016-11-24.

[14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[16] Richard Maclin and David W. Opitz. Popular ensemble methods: An empirical study. *Journal Of Artificial Intelligence Research*, 11:169–198, 1999.

[17] Dan Claudiu Cireşan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. Deep, big, simple neural nets for handwritten digit recognition. *Neural Computation*, 22(22):3207 – 3220, 2010.

[18] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. 2012 special issue: Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 32(Selected Papers from IJCNN 2011):323 – 332, 2012.

[19] A. Ruta, Li Yongmin, and Liu Xiaohui. Real-time traffic sign recognition from video by class-specific discriminative features. *Pattern Recognition*, 43(1):416 – 430, 2010.

[20] A. Ellahyani, M. El Ansari, and I. El Jaafari. Traffic sign detection and recognition based on random forests. *Applied Soft Computing*, 46:805 – 815, 2016.

[21] F. Zaklouta and B. Stanciulescu. Real-time traffic sign recognition in three stages. *Robotics and Autonomous Systems*, 2012.

[22] S. Maldonado-Bascon, S. Lafuente-Arroyo, P. Gil-Jimenez, H. Gomez-Moreno, and F. Lopez-Ferreras. Road-sign detection and recognition based on support vector machines. *IEEE Transactions on Intelligent Transportation Systems*, 8(2):264 – 278, 2007.

[23] D. Cires¸an, U. Meier, J. Masci, and J. Schmidhuber. Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32:333 – 338, 2012.

[24] Volvo Group. Sign display with road sign information. http://support.volvocars.com/uk/cars/Pages/owners-manual.aspx?mc=v526hbat&my=2016&sw=15w46&article=7e283a448975f5bdc0a80151745da716, 2016. Accessed: 2016-11-24.

[25] Volkswagen Group. Sign assist:driving:volkswagen uk. http://www.volkswagen.co.uk/technology/driving/sign-assist, 2016. Accessed: 2016-11-24.

[26] Mercedes-Benz. Merceds-benz techcenter: Traffic sign assist. http://techcenter.mercedes-benz.com/en/traffic_sign_assist/detail.html, 2016. Accessed: 2016-11-24.

[27] Jonathan Vanian. Tesla isn't the only company with bold plans for self-driving cars. *Fortune.com*, page 1, 2016.

[28] Don Reisinger. 10 automakers, tech firms researching self-driving cars. *eWeek*, page 1, 2015.

[29] William Wong. The automotive supercomputer. *Electronic Design*, 64(4):16 – 20, 2016.

[30] Pascale-L. Blyth, Milos N. Mladenovic, Bonnie A. Nardi, Hamid R. Ekbia, and Norman M. Su. Expanding the design horizon for self-driving vehicles: Distributing benefits and burdens. *IEEE Technology & Society Magazine*, 35(3):44 – 49, 2016.

[31] Roman Zakharenko. Self-driving cars will change cities. *Regional Science and Urban Economics*, 2016.

[32] Matt Vella and Katy Steinmetz. The increasingly compelling case for why you shouldn't be allowed to drive. (cover story). *Time*, 187(8):52 – 57, 2016.

[33] Nathan A. Greenblatt. Self-driving cars and the law. *IEEE Spectrum*, 53(2):46 – 51, 2016.

[34] Nick Belay. Robot ethics and self-driving cars: How ethical determinations in software will require a new legal framework. *Journal of the Legal Profession*, 40(1):119 – 130, 2015.

[35] Jeffrey R. Zohn. When robots attack: How should the law handle self-driving cars that cause damages. *Journal of Law, Technology & Policy*, 2015(2):461, 2015.

[36] Self-driving cars: An ethical perspective. *Penn Bioethics Journal*, 11(2):8, 2015.

[37] Lit-Fui Lau and Michael A. Brodney. *Alzheimer's Disease. [Elektronisk resurs]*. Topics in Medicinal Chemistry: 2. Berlin, Heidelberg : Springer-Verlag Berlin Heidelberg, 2008., 2008.

[38] World health organization. Dementia: A public health priority, 2012.

[39] Early detection of alzheimer's disease using structural mri: A research idea. *Life Science Journal-Acta Zhengzhou University Overseas Edition*, (3):1072, 2012.

[40] James Bergstra, Frédéric Bastien, Olivier Breuleux, Pascal Lamblin, Razvan Pascanu, Olivier Delalleau, Guillaume Desjardins, David Warde-Farley, Ian Goodfellow, Arnaud Bergeron, et al. Theano: Deep learning on gpus with python. Citeseer, 2011.

[41] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

[42] Fei-Fei Li, Andrej Karpathy, and Justin Johnson. Lecture notes in cs231n convolutional neural networks for visual recognition: Lecture 1 cnns in practice. `http://cs231n.stanford.edu/slides/winter1516_lecture12.pdf`, February 2016. Accessed: 2016-11-24.

[43] Khronos group. Khronos products. `https://www.khronos.org/conformance/adopters/conformant-products#opencl`, 2016. Accessed: 2016-11-24.

[44] Fei-Fei Li, Andrej Karpathy, and Justin Johnson. Lecture notes in cs231n convolutional neural networks for visual recognition: Lecture 1 cnns in practice. `http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf`, February 2016. Accessed: 2016-11-24.

[45] Intel. How intel® xeon phi™ processors benefit machine learning/deep learning apps and frameworks. https://software.intel.com/en-us/blogs/2016/06/20/how-xeon-phi-processors-benefit-machine-and-deep-learning-apps-frameworks, 2016. Accessed: 2017-01-05.

[46] Blair Hanley Frank. Google's new chip makes machine learning way faster. http://www.computerworld.com/article/3072652/cloud-computing/googles-new-chip-makes-machine-learning-way-faster.html, 2016. Accessed: 2017-01-05.

[47] AMD. Amd introduces radeon instinct: Accelerating machine intelligence. http://www.amd.com/en-us/press-releases/Pages/radeon-instinct-2016dec12.aspx, 2016. Accessed: 2017-01-05.

[48] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In *IEEE International Joint Conference on Neural Networks*, pages 1453–1460, 2011.

[49] Sander Dieleman, Colin Raffel, Eben Olson, Jan Schlüter, and Søren Kaae Sønderby. Lasagne. https://github.com/Lasagne/Lasagne, 2014. Accessed: 2016-10-30.

[50] Dan C. Ciresan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. *CoRR*, abs/1202.2745, 2012.