



Norwegian University of  
Science and Technology

# End-to-end steering angle prediction and object detection using convolutional neural networks

**Øyvind Kjeldstad Grimnes**

Master of Science in Computer Science

Submission date: June 2017

Supervisor: Frank Lindseth, IDI

Norwegian University of Science and Technology  
Department of Computer Science



# Abstract

**Introduction** Artificial neural networks have enabled a new generation of autonomous vehicles. Public datasets can possibly be used to build networks capable of predicting steering angles and detect objects for self-driving vehicles.

**Objectives** The objective of this paper is to investigate whether steering angles and object bounding boxes can be predicted from a single image input. A novel architecture combining both models into a steering angle predictor with incorporated object detection is also explored.

**Method** An object detection model was created based on modern bounding box regression networks. It was trained to detect cars, trucks, traffic lights, pedestrians, bikers, and traffic signs with data from German Traffic Sign Detection Challenge and Udacity. A steering angle prediction network, which was trained using driving data provided by Udacity, was created. Both models shared pretrained layers from VGG16 for feature extraction. The models were also combined into a single model that aimed to make the steering angle predictor more robust by providing explicit object detections as additional inputs.

**Results** The detector achieved a mean average precision of 44.02 on the validation dataset, which was a combination of images from both the German Traffic

Sign Detection Challenge and Udacity datasets. Training the detector on multiple datasets with similar input data, but different annotations reduced the models performance. The steering angle predictor achieved a root mean squared error of 0.0645. By incorporating object detection into the predictor model, the error rose to 0.0653.

**Conclusion** It is possible use public datasets to build models that are capable of predicting steering angles and detect objects from images. Incorporating an object detector into the steering angle predictor did not improve its performance.

# Norsk Sammendrag

**Introduksjon** Kunstige nevrale nettverk har muliggjort en ny generasjon autonome kjøretøy. Offentlige datasett kan potensielt brukes til å bygge nettverk som kan predikere styringsvinkler og lokalisere objekter for selvkjørende biler.

**Mål** Målet for denne oppgaven var å undersøke om man kan predikere styringsvinkler og objekt lokasjoner basert på bilder fra offentlige datasett. En ny arkitektur som kombinerte styringsvinkelpredikatoren og objektlokaliseringen ble også testet.

**Metode** An modell som kunne detektere objekter ble konstruert basert på moderne lokaliseringssmodeller. Den ble trent for å detektere biler, lastebiler, trafikklys, fotgjengere, syklister og trafikkskilt med data fra *German Traffic Sign Detection Challenge* og Udacity. En styringsvinkelpredikator, som ble trent med data fra Udacity, ble også konstruert. Begge modellene delte forhåndstrente lag fra VGG-16. Styringsvinkelpredikatoren ble også modifisert til å bruke objektlokaliseringens prediksjoner som inndata i tillegg til bildet.

**Resultater** Objektlokaliseringssmodellen oppnådde en *mean average precision* på 44.02 på testdataene, som bestod av bilder fra både *German Traffic Sign Detection Challenge* og Udacity. Ytelsen ble dårligere av å trenere på flere datasett med lignende bilder, men ulike annotasjoner. Styringsvinkelpre-

dikatoren oppnådde en *root mean squared error* på 0.0646. Med integrert objektlokalisering økte feilen til 0.0653.

**Konklusjon** Det er mulig å bruke offentlige dataset til å bygge modeller som kan predikere styringsvinkler og detektere objekter basert på bilder. Å integrere objektlokalisering i modellen som predikerte styringsvinkler forbedret ikke ytelsen.

# Preface

This report is the Master's thesis of the author, and it concludes his study in Computer Science at the Department of Computer Science (IDI) at the Norwegian University of Science and Technology (NTNU). This report was written in the spring of 2017, and was supervised by Frank Lindseth.

The thesis is related to the fields of artificial intelligence and visual object localization. The reader is assumed to have basic knowledge of both computer vision and artificial neural networks (ANNs).

A handwritten signature in black ink, appearing to read "Øyvind Grimnes". The signature is fluid and cursive, with a horizontal line underneath it.

Øyvind Grimnes

Trondheim, June, 2017



# Contents

Abstract . . . . .	I
Norsk Sammendrag . . . . .	III
Preface . . . . .	V
<b>1 Introduction</b>	<b>1</b>
1.1 Research Motivation . . . . .	1
1.2 Research Topic and Questions . . . . .	2
1.3 Requirements . . . . .	3
1.4 Report Outline . . . . .	5
<b>2 Background</b>	<b>7</b>
2.1 Hardware . . . . .	7
2.1.1 Parallelized computations in graphics processing units (GPUs)	7
2.2 Software . . . . .	8
2.2.1 Compute Unified Device Architecture (CUDA) . . . . .	8
2.2.2 Tensorflow and Keras . . . . .	8
2.2.3 Robot Operating System (ROS) . . . . .	9
2.3 Data . . . . .	10
2.3.1 German Traffic Sign Detection Challenge (GTSDC) . . . . .	10
2.3.2 Udacity . . . . .	11
2.4 Deep learning . . . . .	12
2.4.1 Activation . . . . .	12
2.4.2 Normalization . . . . .	14
2.4.3 Optimization . . . . .	14
2.5 Deep learning in computer vision . . . . .	15
2.5.1 Object detection . . . . .	15
2.6 Steering angle prediction . . . . .	17
2.7 Transfer learning . . . . .	17

<b>3 Methodology</b>	<b>19</b>
3.1 Experimental setup . . . . .	19
3.1.1 Implementation . . . . .	19
3.1.2 Evaluation . . . . .	19
3.2 Feature extractor . . . . .	20
3.2.1 Choosing a pretrained model . . . . .	20
3.2.2 Choosing pretrained layers . . . . .	22
3.3 The object detector . . . . .	25
3.3.1 Output structure . . . . .	26
3.3.2 Architecture . . . . .	28
3.3.3 Training . . . . .	31
3.4 The steering angle predictor . . . . .	35
3.4.1 Architecture . . . . .	35
3.4.2 Training . . . . .	37
3.5 Steering angle predictor with incorporated object detection . . . . .	38
3.5.1 Architecture . . . . .	39
3.5.2 Training . . . . .	41
3.6 Summary . . . . .	41
<b>4 Results and Discussion</b>	<b>43</b>
4.1 Results . . . . .	43
4.1.1 The object detector . . . . .	43
4.1.2 The steering angle predictor . . . . .	47
4.1.3 The steering angle predictor with incorporated object detection . . . . .	49
4.2 Analysis . . . . .	50
4.2.1 The object detector . . . . .	50
4.2.2 The steering angle predictor . . . . .	55
4.2.3 The steering angle predictor with incorporated object detection . . . . .	56
4.3 Summary . . . . .	57
<b>5 Conclusions and future work</b>	<b>59</b>
5.1 Conclusions . . . . .	59
5.2 Future Work . . . . .	60
5.2.1 End-to-end driver . . . . .	60
5.2.2 Converting problems to a decision problems for pre-training . . . . .	61
5.2.3 Adversarial training . . . . .	61
5.2.4 Working with time . . . . .	62



X

# List of Figures

1.1	A diagram showing the architecture of the system . . . . .	5
2.1	The distribution of classes in the traffic sign detection datasets from German Traffic Sign Detection Challenge (GTSDC). . . . .	11
2.2	The distribution of classes in the object detection Udacity datasets.	12
2.3	Rectified Linear Unit, Sigmoid, and tanh. The plot was created by Vanessa Imiloa ( <a href="https://imiloainf.wordpress.com/2013/11/06/rectifier-nonlinearity/">https://imiloainf.wordpress.com/2013/11/06/rectifier-nonlinearity/</a> ) . . . . .	13
3.1	The size, complexity and accuracy of popular ANN architectures .	21
3.2	The input image used to illustrate the internal activations in VGG-16. . . . .	23
3.3	Internal activations in VGG-16 after the first block of two convolutional layers and a pooling layer. . . . .	23
3.4	Internal activations in VGG-16 after the second block convolutional and pooling layers. . . . .	24
3.5	Internal activations in VGG-16 after the third block of convolutional and pooling layers. . . . .	24
3.6	Internal activations in VGG-16 after the fourth and fifth block of convolutional and pooling layers respectively. . . . .	25
3.7	The detector anchor boxes used in this experiment . . . . .	26
3.8	The 9x9 predictions for a 300x300 input image. . . . .	31
3.9	Augmented image brightness to improve the detector's robustness to different light conditions. . . . .	32
3.10	Augmented image saturation to improve the detector's robustness to color variations. . . . .	32
3.11	Shifted images to improve the dataset's variance . . . . .	33
3.12	Noisy images improves the model's robustness to noise . . . . .	33

3.13	Multiple images generated from a single image by applying random augmentations . . . . .	34
3.14	A diagram showing the architecture of the steering angle predictor with integrated object detection . . . . .	39
4.1	The system successfully detects the most vehicles in daylight . . . . .	44
4.2	The system fails to detect the silhouette of cars in direct sunlight . . . . .	45
4.3	For groups of small objects, the system predicts an average bounding box. . . . .	46
4.4	Predicted bounding boxes for traffic lights are inaccurate. . . . .	46
4.5	The detector needs traffic signs to be at a short distance to be detected. . . . .	47
4.6	The training loss and validation loss of the steering angle predictor. . . . .	48
4.7	The training loss and validation loss of the steering angle predictor with and without integrated object detection respectively. . . . .	49

# List of Tables

3.1	The architecture of the detector. The layers of both the bounding box regressor and classifier are connected to the last shared layer.	30
3.2	The architecture of the steering angle predictor.	36
3.3	The architecture of the steering angle predictor with integrated object detection.	40



# Acronyms

**ANN** artificial neural network. 2, 3, 7–10, 12, 14, 15, 20, 21, V, XI, XIII

**CNN** convolutional neural network. 2, 15–17, 45, XIII

**CPU** central processing unit. 7–9, XIII

**CUDA** Compute Unified Device Architecture. 8, VII, XIII

**FPS** frames per second. 16, 43, 47, 49, 50, XIII

**GPU** graphics processing unit. 7–9, 15, 17, 19, 20, 43, 47, 49, 50, VII, XIII

**GTSDC** German Traffic Sign Detection Challenge. 10, 11, 31, 34, 35, 43, 52, I, III, VII, XI, XIII

**GTSRB** German Traffic Sign Recognition Benchmark. XIII

**GTSRC** German Traffic Sign Recognition Challenge. XIII

**IDI** Department of Computer Science. V, XIII

**ILSVRC14** ImageNet Large Scale Visual Recognition Competition 2014. 20–22, 56, XIII

**IoU** intersection over union. 26, 27, 34, 51, 53, XIII

**LiDAR** Light Detection And Ranging. 2, XIII

**mAP** mean average precision. 16, 19, 43, 50, I, III, XIII

- MLP** Multilayer Perceptron. 12, 13, XIII
- MSE** mean squared error. 35, XIII
- NiN** Network in Network. 16, 29, XIII
- NTNU** Norwegian University of Science and Technology. V, XIII
- R-CNN** region proposal convolutional neural network. 15, 16, XIII
- ReLU** Rectified Linear Unit. 13, 14, 29, 30, 36, 40, XI, XIII
- RMSE** root mean squared error. 19, 35, 38, 41, 47, 49, 54, 55, 57, 58, II, IV, XIII
- RNN** recurrent neural network. XIII
- RoI** region of interest. 16, XIII
- ROS** Robot Operating System. 9, VII, XIII
- RPN** region proposal network. 16, XIII
- SSD** Single Shot Detection. 16, 25, 26, 52, XIII
- VOC** Visual Object Challenge. 17, 50, XIII
- YOLO9000** You Only Look Once 9000. 16, 17, 25, 26, 52, XIII

# Chapter 1

## Introduction

### 1.1 Research Motivation

The invention of the steam engine enabled human kind to undertake long journeys and explorations with less effort than ever before. As the engines improved over the ages, their power increased and their cost decreased. Today, vehicles with powerful, highly advanced petrol and electrical motors are available for everyone, and our society is based on the easily accessible transport for goods and people they provide.

As our cars have become more powerful, there have been concerns about the security of people and material inside and outside the car. Security measures such as seat belts and air bags have reduced the number of fatalities in traffic accidents. Reports show that the main cause of accidents are human errors [33]. We have already introduced functionality to our cars that partly override the human driver to prevent potentially dangerous situations, such as anti-lock brakes and traction control. In modern vehicles, systems such as adaptive cruise control and emergency braking have aided the driver for many years already. Even though these systems improves traffic safety, they are merely trying to mitigate

the symptoms of the main problem. The person operating the vehicle.

Over the last decade, many large car manufacturers and technology companies have invested heavily in artificial intelligence research, aiming to create an agent able to operate a vehicle safely, without intervention from a human driver. A familiar example is Tesla. The company was one of the first to deliver cars with an advanced auto pilot. The commercial autonomous driving systems still require a human to monitor the driving, but with the vast amount of driving data collected from modern cars all around the world, a truly autonomous vehicle is no longer a distant dream, but soon to be reality.

There are many ways to implement such a system. There is a jungle of available sensors that can help the car read and interpret its environment, but every sensor adds both cost and complexity to the system. Most modern autonomous vehicles use a combination of cameras, Light Detection And Ranging (LiDAR), and other sensors to create a robust, but expensive system. If a system using only regular, inexpensive cameras managed to yield super human performance, the cost of commercial autonomous driving systems, and the cost of further research, could be reduced.

An important part of driving a car is to steer in the right direction. Computers have been used to estimate the steering angle for many years, but they have been based on multiple steps such as lane line analysis. This paper will present an approach to make an end-to-end steering angle predictor using convolutional neural networks (CNNs) with incorporated object detection.

## 1.2 Research Topic and Questions

The problems investigated in this thesis was to generate steering angles and detect objects in images for autonomous driving. The research topic was

*End-to-end steering angle prediction and object detection*

The goal was to design and implement ANNs that were able to predict steering

angles and object bounding boxes. The input was a single image from a camera mounted on the front of the vehicle. The results were compared to ground truth labels collected by humans.

The model was based on a single, highly generalized feature extractor for image inputs. Both the object detection and steering angle predictor used the output from one or more of the feature extractor's layers to accomplish their tasks. This reduced the training necessary for each component. This feature extractor reuse is closely related to the field of transfer learning. The following research questions were defined:

1. *Can an artificial neural network learn to predict steering angles from images using only public data?*
2. *Can an artificial neural network learn to detect and locate objects in images using only public data?*
3. *Can steering angles predictions be improved by incorporating a pretrained object detector into the prediction model?*

### 1.3 Requirements

One of the goals of this project was to build a model that was able to successfully predict the steering angle of a vehicle in a real world setting. The predictions should only be based on images recorded from a single forward-facing camera.

The second goal, was to build a detector that could detect road users and traffic signs. This can be used to provide detailed information about the vehicle's surroundings to the passengers, or enable further processing such as detecting speed limits and avoiding pedestrians. In this paper, the information detected should contain the location of other vehicles, pedestrians, traffic lights, and traffic signs.

Both models must be able to process their inputs to provide predictions in real-time. Although the system may benefit from more inputs, such as whether

the road surface is wet, or the condition of the tires, only a single image input was provided in this paper.

The system will consist of three main components (see figure 1.1):

**Feature extractor** To reduce the computational complexity, and the system's memory requirements, a single feature extractor will be responsible for extracting useful information from the input images.

**Road user and traffic sign detector** The object detector should use the output from layers of the feature extractor to detect important objects and information in the input image.

**Steering angle predictor** The steering angle predictor should predict steering angles based on the feature extractor's outputs.

In addition to creating the steering angle prediction and road user detection models, a novel architecture combining the two models will be explored. While there is rarely a direct causality between the surrounding objects of a vehicle and its desired steering angles, there are situations when this connection may improve the predictions, such as:

- When overtaking a vehicle, the steering angle predictor must be certain that there is no oncoming traffic
- When there is an obstacle on the road, the object detection can be used for a more robust obstacle avoidance behaviour
- When a traffic sign restricts the expected steering angle predictions, such as *Right turn only*.

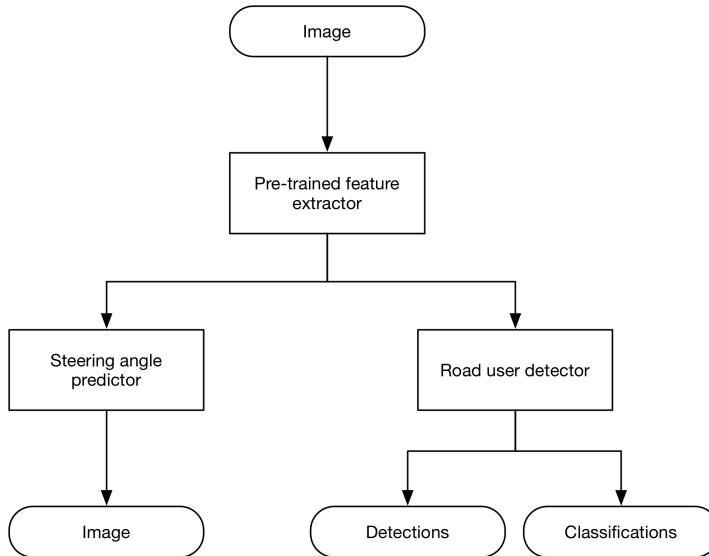


FIGURE 1.1: A diagram showing the architecture of the system

## 1.4 Report Outline

The report contains the following chapters:

**Chapter 1** presents the problem of this thesis, the research motivation and questions.

**Chapter 2** further describes the problem and supplies general background information on the subject.

**Chapter 3** details how the problem is solved, and describes how the system is tested and evaluated.

**Chapter 4** presents and discusses the results of the complete system.

**Chapter 5** summarizes the findings, and what has been achieved.



# Chapter 2

## Background

This chapter will present some of the technologies used in the conducted experiments, a short introduction to some of the techniques used to build the steering angle predictor and object detector, and related work.

### 2.1 Hardware

In the 20th century, one of the main issues with ANNs was that it was computationally expensive to train the networks. It quickly became infeasible to train larger models, and therefore, the domains in which ANNs could be applied were limited. Since then, the hardware in modern computers have greatly improved, adhering to Moore's law [4]. Even more important than building more powerful central processing units (CPUs), was the introduction of the GPU.

#### 2.1.1 Parallelized computations in GPUs

The CPU was designed to be the main processing unit in a computer, which means that it should be able to perform almost any operation sufficiently well. For example reading from and writing to various layers of memory. They typically

contain multiple cores, which help parallelize tasks. In contrast to the CPU, the GPU is a highly specialized component. It is designed to be extremely efficient at parallelizable computations. They may contain hundreds or thousands of cores [22], which enable computation with large matrices while maintaining real-time performance. This enables efficient optimization of ANNs even for very deep architectures. This extraordinary computational power comes at a cost, most noticeably is the speed of reading from and writing to its memory.

## 2.2 Software

In recent years, deep learning have become one of the most popular machine learning techniques due to its ability to model highly complex domains. As it has gained traction, both hardware and software tool kits have been developed to make designing, training, and using deep models more efficient. This section will briefly present some core technologies that have been central to the experiments in this thesis.

### 2.2.1 Compute Unified Device Architecture (CUDA)

Compute Unified Device Architecture (CUDA) is a parallel computing platform created by NVIDIA [21]. The software gives direct access to any CUDA-enabled GPU. While it can be directly interacted with by researchers and developers, it also provides a fixed interface for other tools to interact with the GPU. Thus removing the need for such tools to implement custom interfaces for every model. This have enabled the development of Tensorflow and Keras, which are presented in subsection 2.2.2.

### 2.2.2 Tensorflow and Keras

Tensorflow was originally a machine learning library, but its versatile design have made it a general numerical computation library [31]. It is based on building

computation graphs where nodes are computations, and edges are multidimensional arrays, or tensors. Any computation graph in Tensorflow can be computed on both the CPU and the powerful GPU without any modifications. It was originally developed by one of Google’s machine learning teams, but is now an open source project.

The highly generalized design of Tensorflow makes it applicable to a wide range of problems and domains, but it also makes the syntax for building a ANN quite complex. Keras is a library based on Tensorflow [13], or a similar library called Theano, which provides an interface specialized for ANNs that is more compact, and more readable. This makes experimentation easier, and reduces the time spent writing boilerplate code. Although the library appears simple, it can be customized to be applicable to most problems by designing custom components such as layers or loss functions. It can even be combined with regular Tensorflow implementations as a compact interface for otherwise verbose operations, such as weight and bias creation for layers.

### 2.2.3 Robot Operating System (ROS)

Robot Operating System (ROS) is a collection of tools, libraries and conventions that aims to make the creation of robust behaviour in robots easier in a wide range of robotic platforms. ROSbag was the most important part of ROS’s collection in this thesis. A bag is a container format that stores ROS messages, such as sensory inputs. All the messages are time-coded, which means that a bag of messages can be used to simulate a robot operating in real-time by broadcasting messages according to their time stamp. In this thesis, bags were used to read camera images and steering angles from Udacity’s dataset with driving data. This will be detailed in subsection 2.3.2.

## 2.3 Data

The availability of data is one of the most important aspects to consider when designing an ANN. Large amounts of data is necessary for a model to learn a generalized representation of the problem domain. If a dataset is too small, the model might overfit the available data, and therefore err when interpreting new data. With the rise of big data, the availability of large datasets have surged, and it is now easier to acquire large datasets for optimizing your models. Many companies and organizations have made their datasets publicly available to nurture growth and research in their respective fields. The experiments in this thesis will use two such datasets, as detailed in 2.3.1 and 2.3.2.

### 2.3.1 German Traffic Sign Detection Challenge (GTSDC)

GTSDC was an experiment used to create a benchmark for the current state-of-the-art traffic sign detection algorithms [20]. Teams were encouraged to submit their models, and the submissions were scored and compared. The dataset used to train and test the implementations contained images of roads in Germany with traffic signs. The goal was to locate and classify the traffic signs as precisely as possible. All the data has been made publicly available, and it will be used to teach the system developed in this thesis to detect traffic signs. The distribution of classes in the dataset is illustrated in figure 2.1

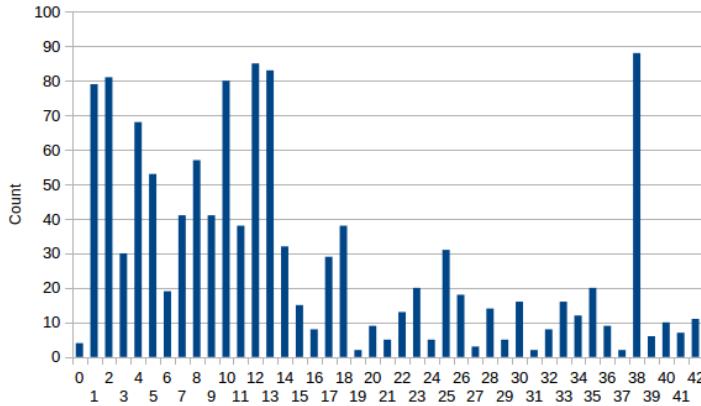


FIGURE 2.1: The distribution of classes in the traffic sign detection datasets from GTSDC.

### 2.3.2 Udacity

As a part of their open source self-driving car project, Udacity have released multiple datasets containing annotations for road users and traffic lights, labels for steering angles, throttle control, along with other metrics recorded while driving.

The dataset containing road user and traffic light annotations with bounding boxes is provided in a basic folder structure containing images from a single forward-facing camera, and labels defining the bounding boxes and classes for objects in the images. The distribution of classes is illustrated in figure 2.2

The driving data is distributed in ROSbag format that was introduced in subsection 2.2.3. The driving data contains time coded images from three forward-facing cameras, steering angles, throttle values, fuel level, and many other metrics. The three cameras can be used to simulate additional training data, as detailed in 3.4.2.

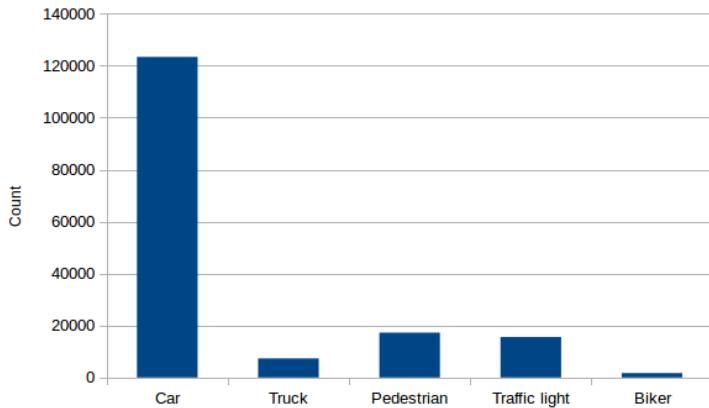


FIGURE 2.2: The distribution of classes in the object detection Udacity datasets.

## 2.4 Deep learning

Since the rebirth of ANNs in the late 1990's, they have become popular since they are highly applicable to wide range of real world problems, because computational power has become less expensive and easily accessible, and because algorithms, such as optimization algorithms [14, 32], have become more effective. This section will introduce some of the techniques used in the experiments in this paper.

### 2.4.1 Activation

When ANNs, or Multilayer Perceptrons (MLPs), were first introduced, they were a series of linear matrix multiplications with an input matrix and a weight matrix for each layer. One of the main issues, as famously claimed by Minsky and Papert [18], was the inability to train networks to solve nonlinear problems. With many other factors, this was one of the reasons the funding and research was cut between the late sixties and early eighties.

An important component of a modern ANN, is the activation function. The output of each node is activated using some nonlinear function to transform the

linear MLP into a general function approximator.

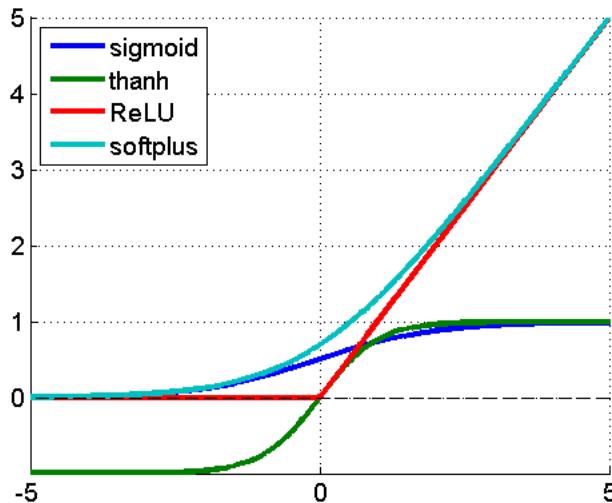


FIGURE 2.3: Rectified Linear Unit, Sigmoid, and tanh. The plot was created by Vanessa Imiloa (<https://imiloainf.wordpress.com/2013/11/06/rectifier-nonlinearity/>)

**ReLU** Rectified Linear Unit (ReLU) [9] appear to be one of the most popular activation functions in recent years. It is a nonlinear function composed of two linear functions, as illustrated in figure 2.3. This means that the gradients used during training will not diminish while being propagated through the network during training. Because the activated output is zero for all negative values, it also promotes sparse networks where few nodes are active at the same time. This prevents the nodes from being dependent on each other.

Some issues with ReLU are that because its derivative is undefined for zero, there are cases where it can be difficult to use, and because nodes are inactive for negative outputs, they can potentially be inactive for all inputs. An activated output of zero will result in a zero gradient, which means that a node that is

inactive for all inputs can never become active again. Those neurons are called dead neurons. To prevent neurons from dying, leaking versions of ReLU are also popular, where negative values are assigned a small fraction of the output value.

**Sigmoid** Sigmoid was one of the earliest activation functions. It scales any input value to the range from zero to one, saturating very small and large values, as seen in figure 2.3. Saturated values results in small gradients, which results in slow training. Therefore, sigmoid is now rarely used as activation function for hidden layers, but still popular as activation function for the output layer if the desired output is binary.

**Tanh** Tanh is also a common activation function. It is also bounded, but can represent both positive and negative values, as illustrated in 2.3. This can potentially make it more expressive compared to sigmoid. One of its main advantages is that its derivative is well defined, which makes it applicable to situations when the ReLU cannot be used.

### 2.4.2 Normalization

To help the model generalize from inputs, it may help to normalize the data. Usually, the input to a network should have a mean value equal to zero, and with a scale of one, depending on the internal design of the model.

Since Ioffe and Szegedy introduced batch normalization as an integrated layer in ANNs [12], it has become a popular way to help models generalize their inputs, and have improved the performance of both existing and new models.

### 2.4.3 Optimization

There have been designed many algorithms for optimizing neural networks. From simple gradient descent, to more complex algorithms such as RMSprop and Adamax [14, 32], they all have theirs strengths and weaknesses.

**Stochastic gradient descend** Stochastic gradient descent is a simple algorithm that updates a network's internal parameters purely based on their approximated gradient. This means that if the gradient is small, optimization will be slow.

**Adamax** Adamax [14] stores additional parameters to improve the optimization performance. It maintains a history of moments and uses them to decide how fast parameters should change. This means that a parameter with a small gradient can still be trained efficiently. In addition, Adamax is claimed to be more stable than other Adam algorithms, and it may be more suitable for sparsely updated parameters.

## 2.5 Deep learning in computer vision

Since the introduction of convolutional layers [15] in ANN architectures, deep learning have rapidly outperformed human designed algorithms and other machine learning approaches in many domains [25]. The depth and complexity of CNNs have rapidly increased [10, 15, 27, 29] as the computational power has increased, especially by exploiting the highly optimized GPU, as discussed in 2.1.1. This has enabled highly accurate classifiers [10, 29], object detectors [17, 23, 24, 26], and image based regressors [1], often matching human performance, and in some cases yielding super human performance [28].

### 2.5.1 Object detection

As the performance of object classifiers have increased, the goal of recognizing objects in images have shifted from simply classifying an object in an image, to both classify and locate multiple objects in larger images [5, 25]. Multiple approaches have been explored, but the most popular architectures seems to be region proposal convolutional neural network (R-CNN) [7, 24] and bounding box regression networks [17, 23, 26].

### Region proposal convolutional neural network

R-CNN is a model based on two separate processes. Initially a region proposal network (RPN) is used to generate regions of interest (RoIs). The generated regions are then cropped and scaled before they are classified using a CNN [8]. While the authors demonstrate a highly accurate system, backed by comparisons to previous state-of-the-art approaches, the architecture was very slow, taking more than 50 seconds per image. Multiple articles have sought to reduce the processing cost by modifying the process of extracting and classifying RoIs [7], and sharing convolution layers between the two networks. Some of the more recent contributions are Faster R-CNN and Mask R-CNN [11, 24], pushing the performance to up to 5 frames per second (FPS) while achieving state-of-the-art accuracy on PASCAL VOC 2007, 2012, and MS COCO.

While 5 FPS is much faster than 50 seconds per frame, it is still far from being a real time object localization model. The front runners for real-time localization at the time of writing, are bounding box regression networks.

### Bounding box regression networks

Unlike R-CNNs, these models strive to both locate and classify objects in images using a single CNN. Sermanet et al. presented Overfeat [26], the performance of this family of architectures have increased in both accuracy and efficiency. Overfeat exploited convolutional layer's invariance to position, and the fact that a convolutional layer with a 1x1 kernel is equivalent to a fully connected layer to simultaneously locate and classify objects in the large images using a single network. This was done without redundant computations in the feature extraction, unlike in a naive sliding window approach or RPNs.

The technique has evolved, and by applying architectural patterns such as Network in Network (NiN) [16], and extracting features from multiple layers, Single Shot Detection (SSD) and You Only Look Once 9000 (YOLO9000) [17, 23] have achieved more than 100 FPS combined with a state-of-the-art mean average

precision (mAP) on the Visual Object Challenge (VOC) 2012 dataset.

YOLO9000 improved the versatility of the original YOLO architecture by training the localization and classification separately. Localization was trained using localization datasets, but the classes used in those datasets are usually higher level synsets of the classes used in classification datasets. Therefore, Redmon and Farhadi also trained the classification using ImageNet [3] and other classification datasets. This enabled the classification to be more discriminative, classifying 9000 non-exclusive classes, while still being able to localize the objects. Connections between different classes were mapped using WordNet [6].

## 2.6 Steering angle prediction

Using CNNs to map raw pixel data to actions are slowly gaining some traction as hardware becomes more powerful. In 2016, NVIDIA released their paper on end-to-end learning for self driving cars [1]. Their system used raw camera inputs to produce steering angle predictions. They demonstrated a robust result where the CNN was able to drive the car on both marked and unmarked roads. This was achieved by at large team of NVIDIA employees with access to expensive equipment such as a car equipped with multiple cameras and sensors for gathering data. This paper will investigate what can be achieved by a single person using only public datasets and limited resources.

## 2.7 Transfer learning

As the complexity of CNNs have increased, so has the time it takes to train them. Training a good convolutional feature extractor may take several weeks [10, 23] on multiple powerful GPUs, and demands huge datasets to avoid overfitting the training data. To mitigate this large overhead when training a new model, transfer learning has become increasingly popular. In deep learning, this entails that a pre-trained model, such as the VGG-16, is reused by copying the architecture

and trained weights into a new model. How many layers of the pre-trained model should be included depend on many factors.

Yosinski et al. [34] show that a convolution layer close to the input layer learns a more general representation than the layers close to the output. When adapting a pretrained model to a new field, this means that the closer the new field resembles the data the original model was trained on, the more layers can be reused. Using VGG-16 as an example, if you were to classify ImageNet images, you could use the complete model as the data used to train the model is the same as you need to classify. On the other hand, if you want to classify one dimensional strings of text, the data would not resemble VGG-16's original training data, and it would be difficult to apply the pre-trained model to the problem.

Appending a convolutional layer to the pretrained model may make adapting to a new problem space easier, and can reduce the need to fine-tune the pretrained weights. This is an advantage when limited data is available, which may cause the pretrained layers to overfit the training data when fine-tuning their weights.

# Chapter 3

## Methodology

### 3.1 Experimental setup

#### 3.1.1 Implementation

The system was implemented using a combination of Tensorflow and Keras. Keras enabled the experimentation necessary to design the system, and Tensorflow provided the performance necessary to optimize and use the models efficiently on the GPU.

#### 3.1.2 Evaluation

The detector and steering angle predictor was evaluated using two measures:

1. A quantitative measure, using mAP for bounding boxes and root mean squared error (RMSE) for steering angles, calculated by predicting previously unseen images from the datasets.
2. Qualitative measure by manually evaluating the performance of the components on an unlabeled dataset collected on urban and suburban roads in

Trondheim.

## 3.2 Feature extractor

Because the system should recognize traffic signs, other motorists, pedestrians, animals, traffic lights and more, the core of the system needed to extract good features for recognizing many different type of objects. Training a custom network from scratch requires a lot of data, and a lot of time. Some of the state-of-the-art deep learning architectures require multiple weeks on high-end GPUs, or even clusters of GPUs. Because there would be little time for experimenting with architectures if they all have to be trained from scratch, it was decided that the core of the system should be based on an existing ANN architecture with pretrained weights.

### 3.2.1 Choosing a pretrained model

In recent years, many accurate models have performed well in object recognition challenges. Because the system should be able to control a car, it did not only need to be accurate, but also efficient enough to provide real-time information.

#### VGG

VGG-19 [27] performed well in ImageNet Large Scale Visual Recognition Competition 2014 (ILSVRC14), but was outperformed by GoogLeNet [29]. It consists of five blocks of convolutional layers with 3x3 kernels followed by max pooling. It was one of the first published architectures not to use kernels larger than 3x3, based on the idea that multiple layers with smaller kernels can replicate a larger kernel's perceptual field. The embeddings from the last convolutional layer are processed by three layers of fully connected layers with 4096, 4096, and 1000 nodes, respectively. This means that the complete model contains more than

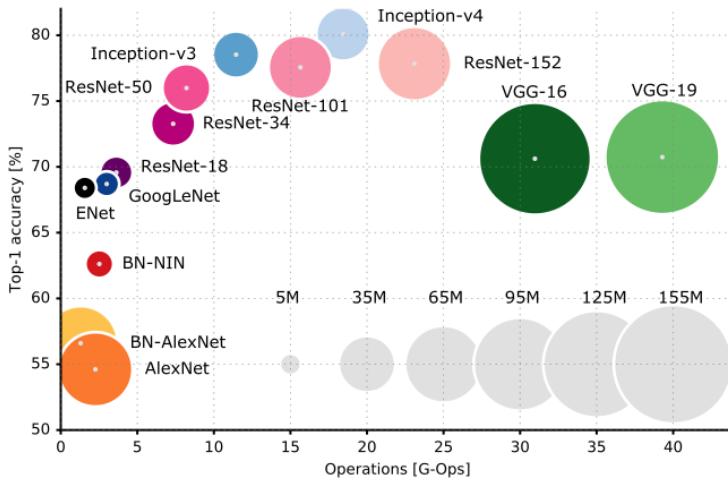


FIGURE 3.1: The size, complexity and accuracy of popular ANN architectures, as presented by Canziani et al. in “An Analysis of Deep Neural Network Models for Practical Applications” [2]

$130e^6$  parameters, making too demanding to run in real-time on most current hardware. The relative size of the VGG models are shown in figure 3.1.

VGG-16 is a more efficient, but less accurate, version of the VGG-19 architecture. According to the authors, it still achieved 25.6 % top-1 error on the ImageNet dataset. Most of the computational complexity is located in the last fully connected layers of the model.

The simple architecture and relatively low complexity in the deeper layers means it is possible to use the deeper layers of a VGG-16 as the pretrained core of the system.

## ResNet

Based on the experimental results showing that more layers generally performs better than larger layers, He et al. designed a one of the deepest models to date. They experimented with models as deep as 56 and 110 layers, and achieved state of the art performance in ILSVRC14. Previously, deeper models had caused the

output to saturate, reducing the models performance. The ResNet architecture solved this issue by adding shortcut connection between layers which enabled the gradients to be passed more effectively through the network during training, and resulted in a more accurate model.

The shortcut connections adds neither extra parameters or computational complexity [10, p. 2], leaving even the 110-layered network is more accurate, smaller, and more efficient than all the VGG models, as illustrated in figure 3.1. ResNet was not designed to work in real time, and the more complex architecture can make it harder to reuse a small subset of the models layers. In addition, the authors claim that the model should be at least 16 layers deep for the shortcut connections to have a positive effect on the model.

Because the system to be built in this paper requires real-time performance, using the number of layers required by residual learning is impractical. Therefore, the deep residual learning model will not be used as the core model in the system.

### 3.2.2 Choosing pretrained layers

When transferring pretrained models from one domain to another, it is important to consider how the pretrained model was trained. If a small dataset was used to train the model, it has most likely overfitted the data to some degree, making it hard to apply the same model in other domains. To avoid this, all the models evaluated in this paper were trained on a subset of the ILSVRC14 dataset containing more than one million images and a thousand unique classes.

It is also important to consider how close the new domain is to the models original domain. If the domains are the same, the whole model can be reused, but if not, one may need to use a deeper layer. The deeper layers are more generic, and can therefore be applied to a wider range of domains. The following paragraphs will illustrate this using the image in figure 3.2 as input.

Figure 3.3 illustrates a sample of the internal activation values of the first pooling layer of VGG-16. It appears that the model initially builds features



FIGURE 3.2: The input image used to illustrate the internal activations in VGG-16.

by detecting horizontal and vertical edges and blobs, similarly to many human designed algorithms. Notice, for example, the high activation values for the number plate in the upper right corner, and that there are still high activation values for the bricks in the background.



FIGURE 3.3: Internal activations in VGG-16 after the first block of two convolutional layers and a pooling layer.

After the second block of convolutional layers, the shape of the car is still recognizable, but each filter have become more specialized. Instead of simply detecting edges, it appears to identify larger features such as the car's outline and its wheels. For most of the filters, the activation values for the background have become weaker. Figure 3.4 shows a sample of the internal activation values of the second pooling layer of VGG-16.

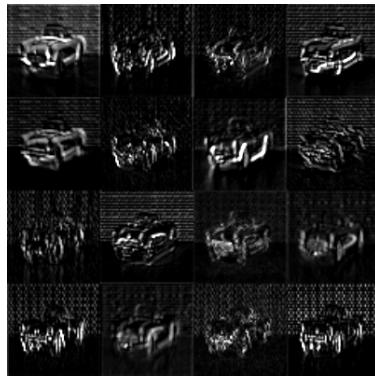


FIGURE 3.4: Internal activations in VGG-16 after the second block convolutional and pooling layers.

Figure 3.5 contains images sampled from the third pooling layer. The filters appear to ignore most of the sections of the image they consider to be background. This means that if the objects in the new domain are considered to be background by the model, for example classifying wall types, this layer may be too shallow to be used in the new domain. Notice how more complex features, such as head lights, have been given large activation values.

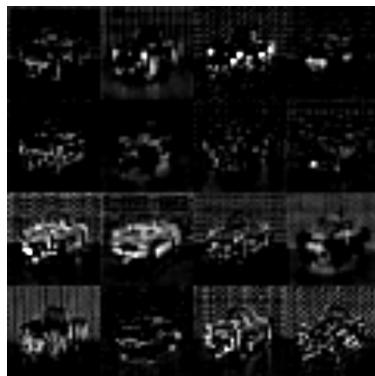


FIGURE 3.5: Internal activations in VGG-16 after the third block of convolutional and pooling layers.

Figure 3.6 shows the activation values of the fourth and fifth pooling layer. These images contains quite specialized information and are difficult to recognize for human beings. Only a small fraction of the fifth layers output is active. This can indicate that the network is sparse, which may be a desirable property to avoid overfitting.

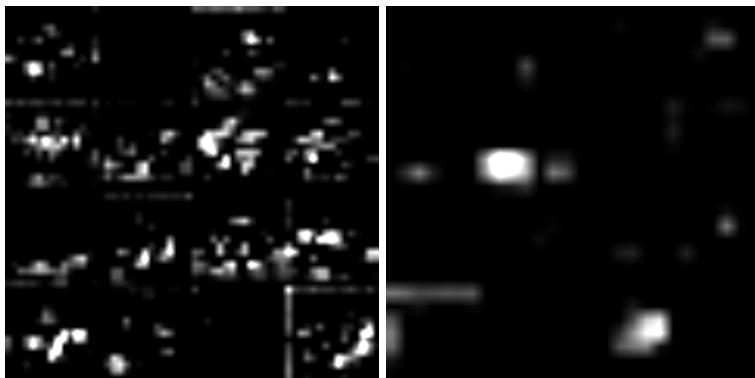


FIGURE 3.6: Internal activations in VGG-16 after the fourth and fifth block of convolutional and pooling layers respectively.

Choosing which layers to use will be based on experiments evaluating the resulting models ability to successfully operate in the new domain.

### 3.3 The object detector

To successfully operate in a real world setting, the system must be able to detect important information on the road. This may include objects such as traffic signs, other vehicles, pedestrians, and traffic lights. To enable real-time performance, the model will be based on bounding box regressor models such as YOLO9000 and SSD.

Unlike those architectures, the classifier output will be completely separated from the bounding box regressor's, making it easier to train the two components individually. This may enable the two components to operate on different synset

levels. For example allowing the bounding box regressor to learn high level synsets such as person, car, traffic sign, and traffic light. At the same time, the classifier can operate on more specialized concepts such as different traffic signs, detecting red, green, and yellow light signals, and different types of cars.

### 3.3.1 Output structure

The output of the detector component is split into bounding box predictions, and classifications.

#### Bounding box regressor

To improve the rate of learning, and possibly improve performance, the regressor did not predict the coordinates of bounding boxes directly. Instead, a set of predefined bounding boxes, hereafter called anchor boxes, were used as the basis for the regressor's predictions. The regressor's predictions were used to translate and scale the anchor boxes, similar to the approach of Liu et al. in their SSD model. The anchor boxes were defined by clustering the bounding box of annotations in the driving dataset released by Udacity containing traffic lights, vehicles, pedestrians and cyclists, similar to how the boxes were selected in YOLO9000. The distance measure used during clustering was 1 - the boxes' intersection over union (IoU).

The anchor boxes used in this article are shown in figure 3.7.

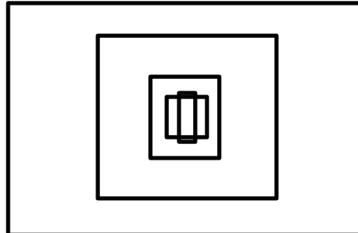


FIGURE 3.7: The detector anchor boxes used in this experiment

For each bounding box, five values were predicted:

**Translation** The two first values are used to translate the image along the x and y axis from the active image fragment's center coordinates. Because the regressor operates using local information, it does not make sense that it should be able to position boxes anywhere in the image. Therefore, the translations were limited to a small radius around the center point of the observed image fragment. In this implementation, that radius was defined as 24 pixels. This way also have improved the model's stability in the early stages of training.

**Scaling** The third and fourth value predicts the scale transformations to be applied to the anchor box to match the object's bounding box. To improve the stability of the model, and better exploit the explicit information provided by the anchor boxes, the scaling in both dimensions were limited to the range from 1 to e.

**Confidence** When combining the first four values with an anchor box, they define a bounding box, but they provide no indication of the models confidence in the prediction. Because the detector and classifier are separated, we cannot use the classifier's confidence to threshold the predicted bounding boxes. The fifth value remedies this by attempting to predict the transformed anchor box' IoU with any objects in the image. This is used to evaluate the models confidence in the predicted box.

To enable the component to detect objects with different shapes and sizes, multiple anchor boxes were be used. The model provided the five output values described for each anchor box. In this implementation, five anchor boxes were used, making the output of the model 25 values for each image fragment.

The formulas used to convert predictions and anchor boxes to the final bounding boxes are shown in equations 3.1 through 3.4. The translation predictions were scaled by the maximum offset radius to avoid the most saturated ranges of the sigmoid function, as seen in equations 3.1 and 3.2. The detector should typically

predict multiple boxes for each object. The number of boxes were reduced using non-maximum suppression [19].

$fx, fy$  = center point of the image fragment predicting the labels

$aw, ah$  = size of the anchor box

$px, py, pw, ph$  = predicted values

$$bx = fx + \sigma^{-1}(px) \times \text{max offset radius} \quad (3.1)$$

$$by = fy + \sigma^{-1}(py) \times \text{max offset radius} \quad (3.2)$$

$$bw = aw \times e^{pw} \quad (3.3)$$

$$bh = ah \times e^{ph} \quad (3.4)$$

## Classifier

The classifier predicted the classes using a vector of confidence values for each class, similar to a traditional classifier. Unlike most classifiers, the model produced multiple such predictions for each image fragment. One for each predicted bounding box.

### 3.3.2 Architecture

#### Shared layers

To reduce the training required, the deepest layers in the detector were the first three blocks of convolutional layers in VGG-16. Those layers were not modified, ensuring that the system's core layers did not overfit the object detector's domain.

This was important as the same layers were also used in the steering angle predictor.

Two blocks of convolutional layers were appended the VGG-16's third max pooling layer, as shown in table 3.1. The blocks were inspired by the NiN architecture [16], all convolutional layers were activated using ReLU [9], and normalized using batch normalization [12]. These blocks should adapt the extracted features to the detector's domain, and reduce the size of the output, enabling accurate predictions and a reasonable number of predicted image fragments for both the bounding box regressor and the classifier. Dropout and batch normalization were used to help the model generalize.

A single shared convolutional layer with 512 filters and a 1x1 kernel was appended to the last shared max pooling layer.

### Bounding box regressor layers

The bounding box regressor's output was created by appending a convolutional layer with a 1x1 kernel and 25 filters, five for each anchor box, to the last shared layer, acting like a traditional fully connected layer. The layer's output was activated using the sigmoid function.

### Classifier layers

The classifier's output was created by appending a convolutional layer with a 1x1 kernel and 30 filters, six for each anchor box, to the last shared layer, acting like a traditional fully connected layer. The layer's output was activated using the sigmoid function.

### Output

Because the system's input size was 300x300, the model yielded 9x9 predictions, where each prediction contained five bounding boxes with accompanying confidence

Component	Layer type	Features	Kernel	Strides	Activation
VGG-16	Conv2D	64	3x3	1x1	ReLU
	Conv2D	64	3x3	1x1	ReLU
	MaxPool2D	64	2x2	2x2	-
	Conv2D	128	3x3	1x1	ReLU
	Conv2D	128	3x3	1x1	ReLU
	MaxPool2D	128	2x2	2x2	-
	Conv2D	256	3x3	1x1	ReLU
	Conv2D	256	3x3	1x1	ReLU
	Conv2D	256	3x3	1x1	ReLU
	MaxPool2D	256	2x2	2x2	-
Shared	Dropout	256	-	-	-
	Conv2D	512	3x3	1x1	ReLU
	BatchNormalization	512	-	-	-
	Conv2D	256	1x1	1x1	ReLU
	BatchNormalization	256	-	-	-
	Conv2D	512	3x3	1x1	ReLU
	BatchNormalization	512	-	-	-
	MaxPool2D	512	2x2	2x2	-
	Dropout	512	-	-	-
	Conv2D	1024	3x3	1x1	ReLU
	BatchNormalization	1024	-	-	-
	Conv2D	512	1x1	1x1	ReLU
	BatchNormalization	512	-	-	-
	Conv2D	1024	3x3	1x1	ReLU
Regressor	BatchNormalization	1024	-	-	-
	MaxPool2D	1024	2x2	2x2	-
	Conv2D	512	1x1	1x1	ReLU
Classifier	Conv2D	25	1x1	1x1	Sigmoid
Classifier	Conv2D	36	1x1	1x1	Sigmoid

TABLE 3.1: The architecture of the detector. The layers of both the bounding box regressor and classifier are connected to the last shared layer.

values and classifications. Figure 3.8 illustrates the output resolution. The actual receptive field for each prediction is larger than the squares in the figure.

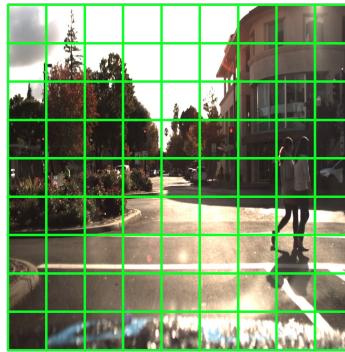


FIGURE 3.8: The 9x9 predictions for a 300x300 input image.

### 3.3.3 Training

#### Data

The bounding box regressor was trained in a supervised fashion on a combination of the GTSDC dataset and the annotated driving data made public by Udacity. Both datasets are described in 2.3.

#### Data augmentation and preprocessing

Because the model was required to operate in a variety of weather and lighting conditions, as encountered in the real world, aggressive augmentation was applied to the input data. This also helped mitigate the model's tendency to overfit the relatively small collections of training data.

**Brightness** To simulate over exposed and under exposed images the brightness of the input was modified. The image's original brightness was multiplied by a factor sampled from a normal distribution with mean equal to one, and a standard deviation of 0.7, based on experimentation. This is illustrated in figure 3.9.



FIGURE 3.9: Augmented image brightness to improve the detector's robustness to different light conditions.

**Saturation** To make the model more robust to variations in color, the saturation of the input images was multiplied by a factor from a normal distribution with mean equal to one, and a standard deviation of 0.8, based on experimentation. This is illustrated in figure 3.10. This may have helped the model recognize objects such as traffic signs with faded colors, and cars with similar color tones.

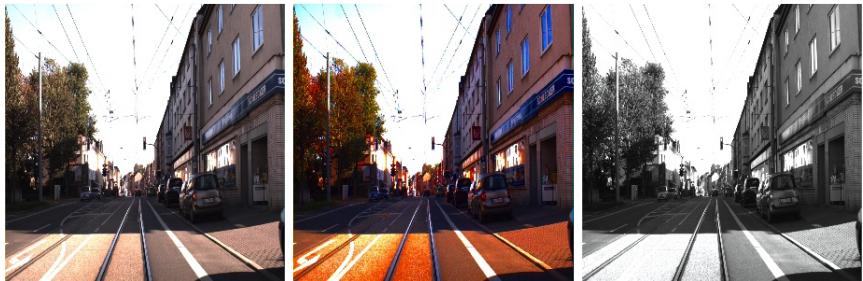


FIGURE 3.10: Augmented image saturation to improve the detector's robustness to color variations.

**Shift** To make the model more robust to minor translations, the input images were shifted along both axes. The offset was sampled from a normal distribution with mean equal to zero and a standard deviation of 20, based on experimentation. The fill color was randomized. This is illustrated in figure 3.11.



FIGURE 3.11: Shifted images to improve the dataset's variance

**Noise** To increase the robustness of the model [30], and force it to learn better features, a lot of noise was added to the input images. The noise was sampled from the normal distribution and multiplied with the image's pixel values. This is illustrated in figure 3.12.



FIGURE 3.12: Noisy images improves the model's robustness to noise

When randomly combining all the augmentations, the result can vary greatly for a single input image. This effectively created a larger dataset from the provided training data, and helped the model to become more robust. Some examples of augmented images are illustrated in figure 3.13. Training samples were procedurally generated from the original training data during training, to save disk space. All images were deallocated after they were processed by the models to save memory.

The input pixel values were centered by subtracting the mean values used in

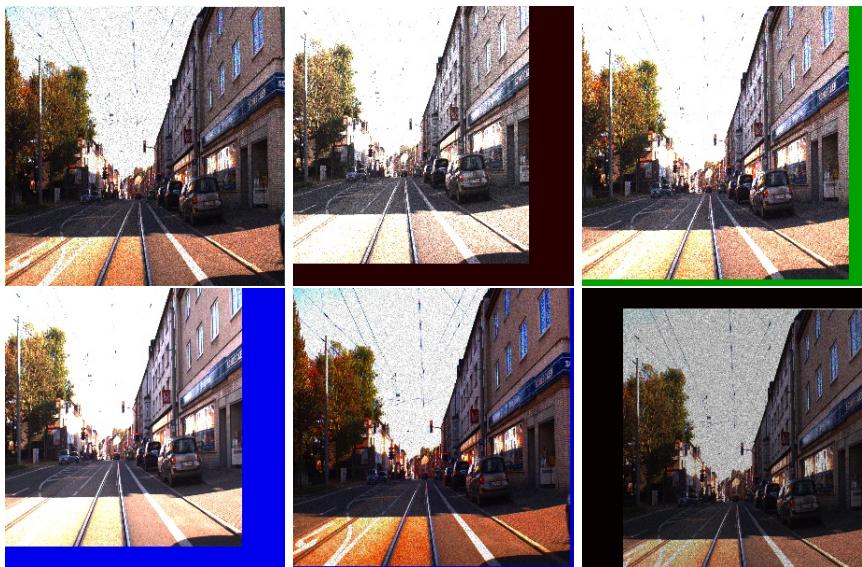


FIGURE 3.13: Multiple images generated from a single image by applying random augmentations

the original VGG paper, and converted to the BRG color space.

### Optimization

When looking at images from the detection datasets, it is obvious that most regions of an image do not contain any interesting objects, and some images contain no interesting objects at all. This unbalance in the input data may have a negative effect on the learning rate of the model. To remedy this, the error from any prediction where the expected IoU is equal to zero was masked.

To enable the system to train on multiple datasets with different annotated object classes, the error of predictions with low expected IoU for all ground truth labels was masked to zero. In this paper, the training data was collected from two datasets released by Udacity, that contained slightly different annotated classes, in addition to the GTSDC dataset, which contained only annotations for traffic signs. If the error was not masked to zero, the system may correctly predict the

presence of a vehicle an image from GTSDC, but it would interpreted as an error because GTSDC did not contain annotations for vehicles.

The detector was optimized with the Adamax optimizer, using the RMSE loss function. RMSE is a reasonable choice for a regression task, such as predicting bounding boxes, but not for classification tasks. RMSE is closely related to the mean squared error (MSE) loss function, but will yield larger errors in this instance. Because the output space and the target space of the model is between zero and one, it will yield a maximum error of one, but usually smaller. This can potentially reduce the model's rate of learning. Because the square root of a number smaller than one is larger than the number itself, the root can be used to make any error larger, which potentially improves the learning rate. The Adamax optimizer will also help improve the learning rate due to its parameterized learning approach.

## 3.4 The steering angle predictor

This model should process a single input image to predict the vehicle's steering angles.

### 3.4.1 Architecture

The steering angle predictor used four blocks of convolutional layers from VGG-16 to help downsample the input size while maintaining a good collection of features. A custom block of convolutional layers was appended to the fourth VGG-16 pooling layer to build more domain specific features, and then processed by two additional convolutional layers. A convolutional layer with a 4x4 kernel was used to pool the features to a vector. This enabled the network to learn the optimal pooling strategy for predicting steering angles. For example, it may give the values collected from the bottom half of the image a higher weight than the top half because the sky rarely affects how a vehicle should turn. The steering angle

prediction was calculated from the result of this pooling operation using a single convolutional layer. This layer was activated with tanh to enable both positive and negative angle predictions. Dropout and batch normalization were used to help the model generalize. The complete model is illustrated in figure 3.2.

Component	Layer type	Features	Kernel	Strides	Activation
VGG-16	Conv2D	64	3x3	1x1	ReLU
	Conv2D	64	3x3	1x1	ReLU
	MaxPool2D	64	2x2	2x2	-
	Conv2D	128	3x3	1x1	ReLU
	Conv2D	128	3x3	1x1	ReLU
	MaxPool2D	128	2x2	2x2	-
	Conv2D	256	3x3	1x1	ReLU
	Conv2D	256	3x3	1x1	ReLU
	Conv2D	256	3x3	1x1	ReLU
	MaxPool2D	256	2x2	2x2	-
	Conv2D	512	3x3	1x1	ReLU
	Conv2D	512	3x3	1x1	ReLU
	Conv2D	512	3x3	1x1	ReLU
	MaxPool2D	512	2x2	2x2	-
Predictor	Dropout	512	-	-	-
	Conv2D	256	3x3	1x1	ReLU
	BatchNormalization	256	-	-	-
	Conv2D	128	1x1	1x1	ReLU
	BatchNormalization	128	-	-	-
	Conv2D	256	3x3	1x1	ReLU
	BatchNormalization	256	-	-	-
	MaxPool2D	256	2x2	2x2	-
	Conv2D	128	3x3	1x1	ReLU
	BatchNormalization	128	-	-	-
	MaxPool2D	128	2x2	2x2	-
	Conv2D	256	3x3	1x1	ReLU
	BatchNormalization	256	-	-	-
	Conv2D	512	4x4	1x1	ReLU
BatchNormalization	512	-	-	-	-
	Conv2D	1	1x1	1x1	Tanh

TABLE 3.2: The architecture of the steering angle predictor.

### 3.4.2 Training

#### Data

The steering angle predictor was trained using the driving data made public by Udacity through their open source self-driving car project. The data contains images and multiple labels, but only the steering angle labels were used in this experiment. The dataset was described in 2.3.2.

#### Dataset augmentation and preprocessing

To improve the data foundation used during training, the dataset was augmented similarly to the detector, as described in subsection 3.3.3, but some approaches were adapted to the new domain:

**Shift** Instead of shifting the image along both axes, the images were shifted only along the x axis. This was used to simulate a situation where the car is not centered in its lane, and should therefore try to move towards the center. This was done by adding a small constant to the original images steering angle, based on the direction of the translation.

**Scale** The images were also randomly scaled along the y axis to simulate sharper and longer corners. The images were scaled by a factor sampled from a normal distribution with mean one and standard deviation of 0.2, based on experimentation. The original image's steering angle was multiplied by the same factor to match the manipulated image.

**Horizontal flip** To balance the training data, and imitate a larger dataset, images were randomly flipped along the first axis, and the steering angle label was inverted. This should prevent the model from preferring to turn either left or right.

**Multiple cameras** The driving data did not contain images from a single camera, but from three individual cameras positioned on the left, right and center of the car. The left and right cameras were used to simulate additional data where the car was not centered in its lane. A small constant was added or subtracted from the true steering angle to simulate a corrective maneuver.

The input pixel values were centered by subtracting the mean values used in the original VGG paper, and converted to the BRG color space.

### Optimization

The model was optimized with the Adamax optimizer using the RMSE loss function of the same reasons as presented in subsection 3.3.3. It was trained using data released by Udacity containing images and steering angles recorded with a human driver. The feature extractor's and detector's weights were not updated while training the steering angle predictor.

## 3.5 Steering angle predictor with incorporated object detection

Similarly to the standard steering angle predictor, this model predicted steering angles based on a single image input, but it also included the pretrained layers of the road user detector. Because both models were already using VGG-16 as a feature extractor, the only modification needed, was to connect the detector's output to the predictor's input. This is illustrated in figure 3.14.

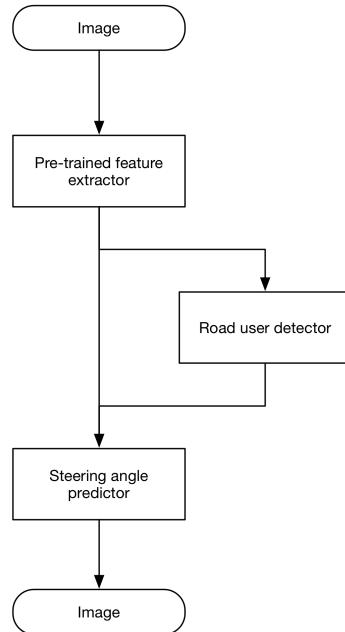


FIGURE 3.14: A diagram showing the architecture of the steering angle predictor with integrated object detection

### 3.5.1 Architecture

The architecture of the combined steering angle predictor was almost identical to the standard predictor, but it concatenated the output of the road user detector with the extracted image features, represented by the *Concatenate* layer in figure 3.3.

Component	Layer type	Features	Kernel	Strides	Activation
VGG-16	Conv2D	64	3x3	1x1	ReLU
	Conv2D	64	3x3	1x1	ReLU
	MaxPool2D	64	2x2	2x2	-
	Conv2D	128	3x3	1x1	ReLU
	Conv2D	128	3x3	1x1	ReLU
	MaxPool2D	128	2x2	2x2	-
	Conv2D	256	3x3	1x1	ReLU
	Conv2D	256	3x3	1x1	ReLU
	Conv2D	256	3x3	1x1	ReLU
	MaxPool2D	256	2x2	2x2	-
	Conv2D	512	3x3	1x1	ReLU
	Conv2D	512	3x3	1x1	ReLU
	Conv2D	512	3x3	1x1	ReLU
	MaxPool2D	512	2x2	2x2	-
Predictor	Dropout	512	-	-	-
	Conv2D	256	3x3	1x1	ReLU
	BatchNormalization	256	-	-	-
	Conv2D	128	1x1	1x1	ReLU
	BatchNormalization	128	-	-	-
	Conv2D	256	3x3	1x1	ReLU
	BatchNormalization	256	-	-	-
	MaxPool2D	256	2x2	2x2	-
	Concatenate	$256 + 25 + 30$	-	-	-
	Conv2D	128	3x3	1x1	ReLU
	BatchNormalization	128	-	-	-
	MaxPool2D	128	2x2	2x2	-
	Conv2D	256	3x3	1x1	ReLU
	BatchNormalization	256	-	-	-
	Conv2D	512	4x4	1x1	ReLU
	BatchNormalization	512	-	-	-
	Conv2D	1	1x1	1x1	Tanh

TABLE 3.3: The architecture of the steering angle predictor with integrated object detection. The concatenate layer merges the image features with bounding box predictions and classifications from the detector.

### 3.5.2 Training

The combined model was trained on the same data, with the same augmentations as the standard steering angle predictor, which was described in section 3.4. It also used the same Adamax optimizer and RMSE loss function. None of the pre-trained feature extractor layers or detector layers were modified during training.

## 3.6 Summary

The experiment was divided into three parts:

1. An object detection model was be implemented using pretrained layers from VGG-16, and it was be trained to detect other road users and traffic signs.
2. A steering angle predictor aimed to predict the correct steering angles from images. It was also be based on pretrained layers from VGG-16.
3. A steering angle predictor with incorporated object detection was be implemented, aiming to make the predictor more robust with explicit information about nearby objects. This was achieved by combining the two other models.

The results of all experiments were evaluated using both a quantitative and qualitative measure.



# Chapter 4

## Results and Discussion

This chapter will present the results of the experiments described in chapter 3. The detector, steering angle predictor, and steering angle predictor with object detection are reviewed separately. All three models will first be evaluated quantitatively, then tested on data captured on the streets of Trondheim for a qualitative evaluation.

### 4.1 Results

#### 4.1.1 The object detector

The object detector was able to operate at a frame rate of 60 FPS on an NVIDIA GTX 1080 GPU.

##### Quantitative measure

The detector was evaluated by calculating the mAP of the predicted bounding boxes on all the test data. It achieved a mAP of 44.91 on the Udacity test data, and 25.73 on the GTSDC test data, which resulted in a total mAP of 44.02.

### Qualitative measure

An example recorded as the sun was setting in Trondheim can be seen by visiting this link: <https://youtu.be/iYsWsLdqtcU>

An example recorded at night can be seen by visiting this link: <https://youtu.be/XouMG3BwJnk>

The generated bounding boxes are mostly matching the detected objects' outline, but they seem to change noticeably between frames. This was not the case when training on dataset with similar annotated classes.

The model did not detect all object types equally successfully. The following paragraphs will describe the performance of the detector for every object in detail.

**Vehicles** The detector seems to successfully predict bounding boxes for most vehicles in daylight, as shown in Figure 4.1, but it fails to recognize cars in some under-exposed images, and it fails to recognize the silhouette of cars. This means that when driving towards the sun, it will not detect cars in front of the camera before they are a few meters away. An example of an image where the detector failed to detect cars due to direct sunlight can be seen in Figure 4.2.



FIGURE 4.1: The system successfully detects the most vehicles in daylight

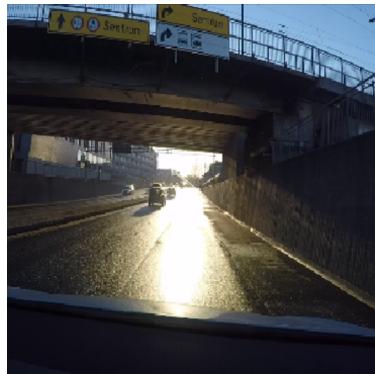


FIGURE 4.2: The system fails to detect the silhouette of cars in direct sunlight

It detects cars of most sizes, ranging from around 20 to 300 pixels in size, and it appears to be robust to the angle of the car. As expected from the properties of CNNs, it can easily detect multiple objects in the same image. The main issue appears when there are multiple small objects grouped together. In that case the detector often predicts a single box that appears to be an approximation of all the objects instead of locating one or more individual objects, as illustrated in figure 4.3. This behaviour also appears when two objects cross paths. The predicted box sometimes slides quickly from one object to the other just before and after the objects overlap each other.

The detector appears to be robust to the relative speed of the objects, correctly predicting bounding boxes for cars crossing the camera's field of view. This could potentially have been an issue as objects moving at high speeds may become blurred when recorded.

**Traffic lights** The detector appears to recognize traffic lights in intersections, but the generated bounding boxes are usually not centered on the object, rather floating nearby. This is illustrated in figure 4.4. It appears to prefer traffic lights approximately a couple of lane widths away, and does not detect traffic lights that are facing away from the vehicle.

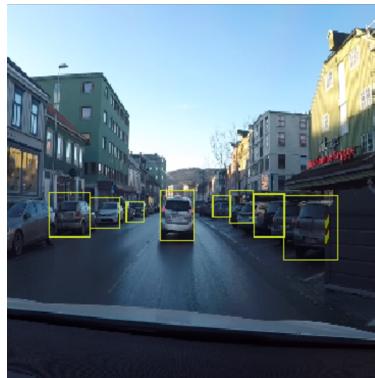


FIGURE 4.3: For groups of small objects, the system predicts an average bounding box. Notice box number three from the left.

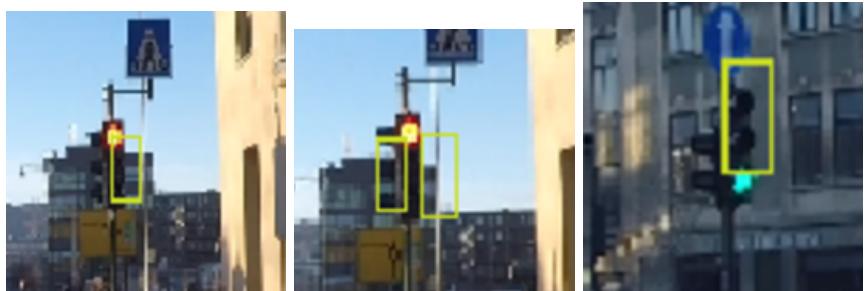


FIGURE 4.4: Predicted bounding boxes for traffic lights are inaccurate.

**Pedestrians** The detector generally fails to detect pedestrians. A bounding box might appear for a frame or two, but never as a consistent indication of a person.

**Traffic signs** The system does detect some traffic signs, but usually at a short distance, as shown in figure 4.5.



FIGURE 4.5: The detector needs traffic signs to be at a short distance to be detected. Notice the bounding box for the sign on the left. The detector correctly recognizes that there is a sign there, but fails to place the bounding box.

### 4.1.2 The steering angle predictor

The predictor was able to operate at a frame rate of 60 FPS on an NVIDIA GTX 1080 GPU, which may qualify for real-time performance.

#### Quantitative measure

The performance of the predicted steering angles were measured using RMSE. Figure 4.6 plots the training and validation error of the predictor. The training error is larger than the validation error. This is probably caused by the augmentations applied to the input and dropout layers.

The final RMSE of the steering angle predictor with object detection was 0.0645. In comparison, predicting zero for all inputs yields a RMSE of 0.32.

#### Qualitative measure

An example in daylight can be seen by visiting this link: [https://youtu.be/k\\_CMFqbeXqk](https://youtu.be/k_CMFqbeXqk)

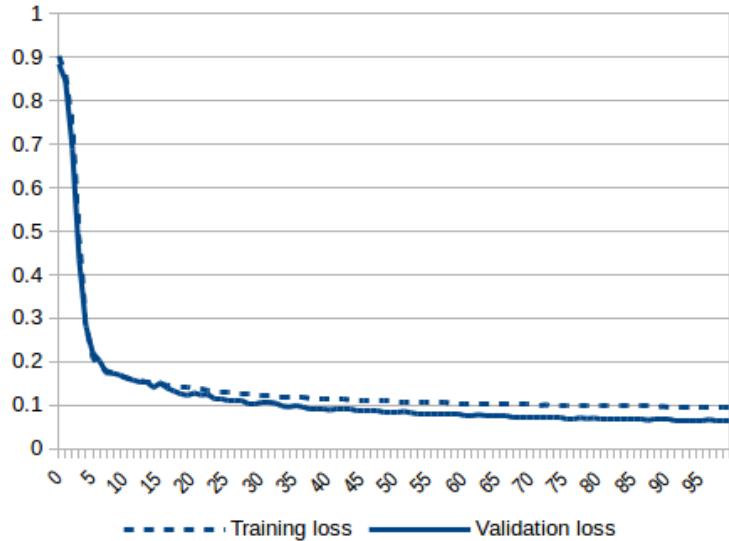


FIGURE 4.6: The training loss and validation loss of the steering angle predictor. The model was optimized for 100 epochs with 200 batches of 16 samples. Multiple training sessions were averaged to produce the data displayed in this graph.

An example recorded at night can be seen by visiting this link: <https://youtu.be/tj0Quuq5q5s>

The driver seems to recognize the curvature of the road. When the road turns, it predicts steering angles that seem to correlate with the sharpness of the turn. It appears to prefer to turn left as all predicted angles are centered on a small negative angle, not on the zero angle. It appears to be relatively robust in situations such as to overtaking cars, driving behind cars, and passing parked cars, but sometimes it desires to steer into oncoming traffic seemingly without any specific reason.

### 4.1.3 The steering angle predictor with incorporated object detection

The object detecting predictor was able to operate at a frame rate of 50 FPS on an NVIDIA GTX 1080 GPU, which may still qualify for real-time performance.

#### Quantitative measure

The performance of the predicted steering angles were measured using RMSE. The final RMSE of the combined steering angle predictor with object detection was 0.0653. The RMSE of the steering angle predictor without object detection was 0.0645. Which means that it is likely that the object detection did not help the model predict better steering angles.

Figure 4.7 plots the training and validation error of the both the steering angle predictors. This illustrates how the explicit object detection information affected, or rather failed to affect, the system.

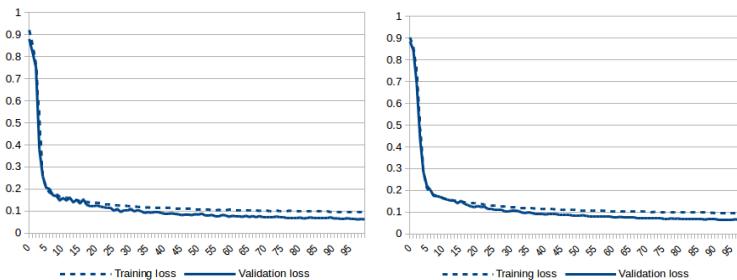


FIGURE 4.7: The training loss and validation loss of the steering angle predictor with and without integrated object detection respectively. The models were optimized for 100 epochs with 200 batches of 16 samples. The results for each graph were generated by averaging the result of multiple iterations.

### Qualitative measure

The behaviour was similar to the standard predictor, but the input from the detector component appeared to confuse the system. When the car met oncoming traffic, or there were cars parked on the side the road, the activations in the detector's output became stronger, and the predicted steering angles became more unstable. When there was no oncoming traffic, and there were no cars parked on the side of the road, the model seemed to perform similar to the standard steering angle predictor.

## 4.2 Analysis

### 4.2.1 The object detector

The detector did achieve real-time performance at 60 FPS, even with the unnecessarily computationally intensive, pre-trained VGG-16 layers. A more powerful GPU would probably yield even better performance. The model achieved a mAP of 40.02 on the test data, which is not very high compared to the state-of-the-art detectors on VOC. This subsection will discuss the result, and how the implementation might can be improved.

#### Robustness

**Brightness** Even though the brightness of input images, among other properties, was heavily augmented, the detector still failed to recognize objects in some dark images, and when only the basic shapes, such as the silhouette, of the objects were visible. All the training data was recorded in daylight, and in almost all the images, the road is correctly exposed. In the test data, the auto exposure of the camera sometimes failed to keep the road correctly exposed. This mismatch between the training data and test data can have affected the performance of the trained model. Using a camera with a high dynamic range might help mitigate

this issue.

**Bounding boxes** The generated bounding boxes' size and position seemed more unstable when training on both the Udacity and German datasets. This is probably caused by the ambiguity introduced when training on multiple datasets, as discussed in 4.2.1.

**Detecting background** Because any prediction with an IoU of zero for all annotations were masked to zero, the model is never shown that it is possible for an image fragment not to contain an object. This can have caused the predicted confidence values to saturate, which can cause false positives to be predicted with high confidence values.

**Heat map** Most false positives appear for a few frames. A heat map could be used to silence any detections until a sufficient number of detections are reported in an area. This adds little complexity, and may remove most of the false detections.

### Detected object types

The detector clearly preferred to locate vehicles even though the training data also contained pedestrians, traffic lights, traffic signs, and bicycles. The following paragraphs will discuss why this happened.

**Representation in datasets** The detector did not detect pedestrians. The object type was annotated in the Udacity datasets, but relatively few images contained pedestrians. During training, the detector appeared to have specialized in vehicle detection at the cost of more sparsely represented object types. This could possibly be prevented by balancing the input data during training, or by weighting the training samples according to their degree of representation in the datasets, as discussed in paragraph 4.2.1

**Inter-class variance** The identifying features of a car are relatively simple: a box with wheels, or a box with a light source on either side, depending on the relative orientation of the car. These basic shapes are easy to identify. A pedestrian may be dressed in black, standing still between dark tree trunks, or next to a traffic light. Or he may be dressed in a huge pink jacket, running across the road, or sitting on a bench. The wide variety of shapes and colors of a person is more difficult to learn than the shape of a car. Combined with the under-representation of pedestrians in the training data, this may have made it difficult for the model to recognize pedestrians.

**Anchor boxes** The over-representation of cars may have made most of the anchor boxes adapt to cars, thus making detection of other object types more challenging for the model. This could possibly be remedied by balancing the annotations used during classification by sampling the same number of annotations from every object type.

**Architecture** Given the relatively small datasets available, using a pretrained model was a reasonable choice, but efficiency and accuracy could probably be improved by creating a custom feature extractor more similar to those of SSD or YOLO9000. The accuracy may also be improved if the system was allowed to modify some of the pre-trained layers.

### Training on multiple datasets

The accuracy of the detector was noticeably reduced when both the Udacity and German traffic signs dataset were used to train it. Event though the model was not directly punished for predicting objects that were not present in the dataset. There could be many reasons for this, such as:

**Ambiguity** Some images contain cars and traffic signs located close to each other. Given an image from the GTSDC dataset that contains both cars and

traffic signs, if an image fragment is located close enough to the traffic sign, it will have an expected IoU greater than zero, so its error will not be masked. If the car was located close to the image fragment, the model may still decide to predict a bounding box for the car instead of the traffic sign. This would result in a large error because the car was not annotated in the input image, even though it would have been a correct prediction in the Udacity datasets. This ambiguity may confuse the network making it less accurate.

This tendency could probably be mitigated if an object detector was trained on the individual datasets, then used to create additional annotations for the other datasets. The new annotations, combined with the annotations from the dataset, would reduce the difference between the training datasets, and possibly improving the performance of the final model.

**Anchor boxes** Because the five anchor boxes were created by clustering the annotations in the training datasets, the anchor boxes may be too generalized to efficiently represent the various objects in the real world. More data means more generalized anchor boxes, which potentially reduce the models performance.

## Optimization

The output of both the bounding box regressor and classifier were activated using the sigmoid function. This saturates the output values, and can result in smaller gradients and reduced learning rate. Although a parameterized optimizer will help prevent this, using a better combination of output activation function and loss function could improve the model. This is further explored in the following paragraphs.

**Output activation** A sigmoid function saturates large positive and negative input values. This means that the loss for a small error in a large box will be small. The same error will yield a larger loss for a medium bounding box, because the predicted values will be in a less saturated range. This is the desired behaviour.

Unfortunately, because negative values are also saturated, the model will produce a small loss for small errors in small bounding boxes. This is not desired behaviour. A small error is much larger relative to a small box than to a large box, which should be reflected by the loss. This may be a reason for why the smaller boxes, which indicates traffic lights and distant objects, are less accurate than the larger boxes.

Using softmax for the classifier's output was not an option because every output vector contained predictions for multiple anchor boxes. Softmax would have strengthened a single class for a single anchor box, and wakened all other classes without considering that the classes were related to different anchor boxes. An alternative would have been to create a custom softmax layer that applied softmax to the different anchor boxes' predictions individually. This would still assume that all the classes for a single anchor box were mutually exclusive, which they were in this experiment, but they may not be if more classes, such as *vehicle*, are introduced.

**Loss function** Using a squared error based loss function is effective when applied to regression tasks, such as predicting bounding boxes, but less effective when used to classify objects. In the detector implementation, a single RMSE loss function was used, and resulted in decent bounding boxes, but inaccurate classifications.

A custom designed loss function could probably improve the model's classification accuracy. A simple approach would be to combine two existing functions, applying RMSE to the bounding box parameters, but a function more fit for classification tasks to the classifier's output. The most common examples are probably categorical and binary cross-entropy. The categorical version assumes that the classes are mutually exclusive, which they were in this experiment, but because a single output vector contained classification for multiple anchor boxes, the loss function should accommodate multiple classes for each vector. Therefore, the binary cross-entropy loss function would be a better choice. Combining it

with the RMSE would yield large errors for the output of both the bounding box regressor and classifier.

**Weighting samples** The number objects contained in the detection datasets was not balanced, as shown in figures 2.2 and 2.1. Therefore, the model was better adapted to detecting the more common objects such as cars than the less common pedestrians. A possible approach to counteract this tendency, without holding back data during training to balance the training process, is to use weighted training samples. The common objects, such as cars would get a smaller weight than pedestrians, preventing the many gradients of car predictions from overpowering the relatively few pedestrian gradients.

**Masking errors** Because the error from predictions not containing object were masked, the model was never taught that a prediction might not contain any objects. This may have caused it to report many false positives, which required a high output confidence threshold. This may prevent the model from detecting objects at a distance, or the silhouettes of cars, as previously identified as a weakness. But, as it was the masking of errors that made training on multiple datasets in this experiment, this effect was unavoidable.

### 4.2.2 The steering angle predictor

The steering angle predictor appeared to predict reasonable steering angles in most cases.

#### Data

This section will discuss how the steering angle predictor may have been affected by the data.

**Unbalanced training data** Roads are mostly straight, this means that the majority of the training data will represent steering angles close to zero. This may

bias the model towards driving straight. To avoid this, the training data should probably be balanced by ignoring or weighting samples with a small ground truth steering angle.

**Preference for turning left** The model appeared to prefer to steer left. This was probably because the model was trained to keep the camera in the center of its lane, but in the test videos, it was often located slightly to the right of the center point. Therefore it continuously tried to make a correcting maneuver towards the center.

## Architecture

The predictions failed to match those of the NVIDIA team. There are many reasons for this, but it may be an indication that the architecture itself was flawed.

**Feature extractor** The detector was connected to the pooling layer of the fourth block of convolutional layers in the VGG-16 model, one block shallower than the detector. It is possible that the features at that point were too specialized to the ImageNet classifications. The model needed to detect features such as lane lines and curbs, but those are features that may usually be considered background in images from the ILSVRC14 dataset.

### 4.2.3 The steering angle predictor with incorporated object detection

The combined predictor did not seem to be robust at all. Under certain conditions, for example when the road was well lit, and there were no other cars, traffic signs, or traffic lights nearby, it appeared to predict reasonable steering angles. The model seemed to be more confused than aided by the extra information provided by the object detector. In theory, it should have been able to achieve at least the

same performance as the steering angles predicted without the extra input from the detector, but it did not.

In addition to the issues described for the standard predictor in subsection 4.2.2, the additional inputs may have caused more issues for the model.

## Data

As discussed in subsection 1.3, the object detector would probably have little effect on most steering angles. Most of the situations where the object detection may be helpful, such as obstacle avoidance, were not present in either the training or the test datasets. Thus, measuring a positive effect for the incorporated object detector proved to be challenging. The result was that the object detecting steering angle predictor performed worse than the original, possibly due to confusion caused by the detector input values.

## Architecture

**Combining image features and detections** The model was provided with three convolutional layers to extract information from the image features and detections with 128, 256, and 512 filters respectively. This may not be enough for the wide variety of inputs possible. Especially considering that the concatenated image features and detection results consisted of 211 features.

## 4.3 Summary

The object detector correctly located most vehicles except from a few edge cases. It appeared to prefer cars, probably due to the overrepresentation of the car class in the training data. Other objects were also detected, but the predicted bounding boxes were less accurate.

The steering angle predictor yielded predictions which appeared to be correct in most situations, and achieved a decent RMSE of 0.0645 on the test data.

The steering angle predictor with integrated object detection did not yield better results than the standard predictor, but achieved an RMSE of 0.0653 on the test data.

# Chapter 5

## Conclusions and future work

### 5.1 Conclusions

An object detection network was based a core of pretrained layers from VGG-16 was implemented. The detector's layers were trained as a regular detector. The detector appeared to be quite stable. It yielded a some false positives that were introduced when the system was trained on multiple datasets with similar images, but different annotations, at the same time. To reduce the effect this had on the model, the errors of false predictions were masked to zero, which helped, but did not make the model as stable as when training on either of the datasets alone. The experiment shows that it should be possible to train a detector using only public datasets.

A steering angle predictor used pretrained layers from VGG-16 for feature extraction. It failed to deliver the stability and accuracy seen in NVIDIA's system, but it appeared to interpret the curvature of roads and predict steering angles that seemed to be correct in most situations. While the experiment suggests that it is possible to train a model to predict steering angles using only public data, it would benefit from more data recorded in a wider range of environments and

conditions.

A novel combination of the object detector and steering angle predictor was implemented. Both components shared feature extraction layers, and the steering angle predictor processed both the image features and the detector predictions to predict steering angles. The model did not perform better than the standard steering angle predictor. It appeared to be more confused than aided by the more explicit information received from the integrated object detector, it was challenging to train robustly, and failed to perform better than regular models such as NVIDIA's.

Multiple approaches for improving the performance of the models were discussed, including revised architectures, generating additional training data, and improved data manipulation.

The approach taken by NVIDIA in their paper on end-to-end self driving cars was proven to be quite effective, and it is probably a better approach than the architectures suggested in this paper.

## 5.2 Future Work

### 5.2.1 End-to-end driver

The novel architecture for steering angle prediction and object detection presented in this paper did not perform well. As described in the introduction, nearby objects rarely affect the steering angles of a car, but they may be more important for the acceleration and braking of a vehicle.

If the steering angle predictor with object detection was extended to also predict throttle and brake controls, the object detection might have a positive effect. For example braking when the car in front is braking, an stopping at stop signs or red lights.

For such a system to work, the model would need to have some notion of the current speed of the vehicle. This could be provided as a separate input,

taught using recurrent layers, or indicated by applying optical flow on the input images. Both the recurrent layers and the optical flow input would also provide information about the speed of other road users, which may be important.

### 5.2.2 Converting problems to a decision problems for pre-training

The amount of available labeled data can make a substantial impact on the performance of a model, but correctly labeled data can be difficult to produce. When predicting steering angles from images, sensors measuring the angle of the steering wheel or tires are needed, and they must be synchronized with images from a camera mounted on the car. This requires hardware that most people do not have at home, making it difficult to gather reliable steering angle data. The data used to train the model will be limited to what is publicly available.

Computational theory shows that while a problem can be hard to solve, such as NP-complete problems, verifying a proposed solution may be considerably easier. Therefore, training a model to verify a proposed prediction might be easier than training a system to predict the actual steering angles. This approach would also reduce the need of labeled data because only the true predictions would need to be labeled correctly. Any false predictions can automatically be generated in the range of the input data for any input image similar to images in the training data.

When constructing a model to solve the original problem, robust, pre-trained layers could be transferred. This approach may be applicable to a wide range of problems.

### 5.2.3 Adversarial training

The same advantages as described in subsection 5.2.2 can also be achieved by not using the decision problem model as pre-training, but rather as the steering angle predictions adversary.

The system would consist two separate networks where one, the predictor, tries to predict the correct steering angle from an input image, and the other, the validator, tries to tell if a provided steering angle is the ground truth, or a prediction from the predictor for a given image.

The networks could be trained in an adversarial manner where:

- The validator attempts to decide whether the steering angle input is the true steering angle for an image input, or one predicted by the predictor.
- The predictor predicts a steering angle for an input image. The prediction error would be based on whether the validator believed it was a ground truth label or not.

These models would be trainable using both labeled and unlabeled data, and the approach may be applicable to a wide range of problems.

#### 5.2.4 Working with time

When detecting objects, it is more likely that predicted bounding box is correct if it has been detected in multiple frames. False positives will typically appear for edge cases in individual frames. If the model knew that there was no prediction in the area of a potential false negative in the previous frames, it may decide not to give it a high confidence value. By maintaining a history, the model could also produce smoother bounding boxes because it knows that objects rarely changes shape, and therefore predicts boxes similar to those predicted in previous frames. It would also enable detection of movement, which can be essential for obstacle detection and avoidance.

Then predicting steering angles, the current steering angle may depend on the steering angles at the previous time steps, and knowing the previous steering angles would allow the model to predict more stable values. The model implemented in this thesis yielded slightly varying angles for each frame, resulting in an

uncomfortable journey, and maybe more importantly, reduced control of the vehicle.

Adding some recurrent layers to the system may give it a notion of time and make it more stable.



# Bibliography

- [1] Mariusz Bojarski et al. “End to End Learning for Self-Driving Cars”. In: *CoRR* abs/1604.07316 (2016).
- [2] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. “An Analysis of Deep Neural Network Models for Practical Applications”. In: *CoRR* abs/1605.07678 (2016).
- [3] J. Deng et al. “ImageNet: A Large-Scale Hierarchical Image Database”. In: *CVPR09*. 2009.
- [4] The Editors of Encyclopædia Britannica. *Moore’s law*. 2017. URL: <https://www.britannica.com/topic/Moores-law> (visited on 05/18/2017).
- [5] M. Everingham et al. *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results*. <http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html>
- [6] Christiane Fellbaum. *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.
- [7] Ross B. Girshick. “Fast R-CNN”. In: *CoRR* abs/1504.08083 (2015).
- [8] Ross B. Girshick et al. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *CoRR* abs/1311.2524 (2013).
- [9] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep Sparse Rectifier Neural Networks”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11)*. Ed. by Geoffrey J. Gordon and David B. Dunson. Vol. 15. Journal of Machine Learning Research - Workshop and Conference Proceedings, 2011, pp. 315–323.
- [10] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *CoRR* abs/1512.03385 (2015).
- [11] Kaiming He et al. “Mask R-CNN”. In: *CoRR* abs/1703.06870 (2017).

- [12] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *CoRR* abs/1502.03167 (2015).
- [13] Keras. *Keras: Deep Learning library for Theano and TensorFlow*. URL: <https://keras.io/> (visited on 05/20/2017).
- [14] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2014).
- [15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105.
- [16] Min Lin, Qiang Chen, and Shuicheng Yan. “Network In Network”. In: *CoRR* abs/1312.4400 (2013).
- [17] Wei Liu et al. “SSD: Single Shot MultiBox Detector”. In: *CoRR* abs/1512.02325 (2015).
- [18] M. Minsky and S. Papert. *Perceptrons*. Cambridge, MA: MIT Press, 1969.
- [19] Alexander Neubeck and Luc Van Gool. “Efficient Non-Maximum Suppression”. In: *Proceedings of the 18th International Conference on Pattern Recognition - Volume 03*. ICPR ’06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 850–855.
- [20] Institut Für Neuroinformatik. *German Traffic Sign Recognition Challenge*. 2011. URL: <http://benchmark.ini.rub.de/?section=gtsrb&subsection=results> (visited on 11/15/2016).
- [21] NVIDIA. *WHAT IS CUDA?* URL: [http://www.nvidia.com/object/cuda\\_home\\_new.html](http://www.nvidia.com/object/cuda_home_new.html) (visited on 05/20/2017).
- [22] NVIDIA. *What’s the Difference Between a CPU and a GPU?* 2009. URL: <https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/> (visited on 05/20/2017).
- [23] Joseph Redmon and Ali Farhadi. “YOLO9000: Better, Faster, Stronger”. In: *CoRR* abs/1612.08242 (2016).
- [24] Shaoqing Ren et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *CoRR* abs/1506.01497 (2015).
- [25] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252.

- [26] Pierre Sermanet et al. “OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks”. In: *CoRR* abs/1312.6229 (2013).
- [27] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *CoRR* abs/1409.1556 (2014).
- [28] J. Stallkamp et al. “Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition”. In: *Neural Networks* 0 (2012), pp. -.
- [29] Christian Szegedy et al. “Going Deeper with Convolutions”. In: *CoRR* abs/1409.4842 (2014).
- [30] Christian Szegedy et al. “Intriguing properties of neural networks”. In: *CoRR* abs/1312.6199 (2013).
- [31] Tensorflow. *About TensorFlow*. URL: <https://www.tensorflow.org/> (visited on 05/20/2017).
- [32] T. Tieleman and G. Hinton. “RMSprop Gradient Optimization”. In: () .
- [33] Statens Vegvesen. “Dybdeanalyser av dødsulykker i vegtrafikken 2015”. In: (2015).
- [34] Jason Yosinski et al. “How transferable are features in deep neural networks?” In: *CoRR* abs/1411.1792 (2014).