Freie Universität Berlin

Bachelor thesis at the Department of Mathematics and Computer Science

Dahlem Center for Machine Learning and Robotics

End-to-End Learning of Steering Wheel Angles for Autonomous Driving

Janis Jendrik Meyer

Matrikelnummer: 4932352

meyerjan@zedat.fu-berlin.de

Betreuer: Prof. Dr. Daniel Göhring

Eingereicht bei: Prof. Dr. Daniel Göhring

Zweitgutachter: Prof. Dr. Dr. (h.c.) habil. Raúl Rojas

Berlin, May 22, 2019

Abstract

Autonomous driving has been an active area of research for many years, especially since the rise of deep learning for computer vision tasks. In the past several years, many different works used end-to-end learning in an attempt to deal with the complexity of the problem, relying on datasets of collected driving data to train their models.

This thesis presents an overview of some of these papers and highlights decisions, considerations and difficulties encountered when creating an autonomous driving model using end-to-end learning. A convolutional neural network is trained to predict steering wheel angles using a dataset of front view images of human driving collected by AutoNOMOS Labs. Different configurations and settings for dataset composition and network input and output are explored and compared. The models are evaluated on a validation subset and with the use of the VisualBackProp algorithm, which visualizes regions of the input with the biggest influence on the network prediction.

The best performing model is able to follow the lane on highways and is shown to have learned to detect and recognize lane markings to be used in the decision-making process.

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

May 22, 2019

Janis Jendrik Meyer

Contents

1 Introduction **1**

2 Related Work **4**

3 Deep Learning **11**
3.1 Neural Networks . . . 11
3.2 Training . . . 12
3.2.1 Gradient Descent . . . 13
3.2.2 Backpropagation . . . 14
3.3 Activation Functions . . . 15
3.4 Loss Function . . . 16
3.5 Convolutional Neural Network . . . 16
3.6 Regularization . . . 18
3.6.1 Dropout . . . 19
3.7 Batch Normalization . . . 19
3.8 VisualBackProp . . . 20

4 Approach **22**
4.1 Dataset . . . 22
4.1.1 Collection . . . 22
4.1.2 Filtering . . . 23
4.1.3 Pruning . . . 24
4.1.4 Augmentation . . . 25
4.2 Preprocessing . . . 26
4.2.1 Rectification . . . 27
4.2.2 Cropping . . . 27
4.2.3 Downsampling . . . 29
4.2.4 Colorspace . . . 29
4.2.5 Normalization . . . 30
4.2.6 Steering Wheel Angles . . . 30
4.3 Network Architecture . . . 31
4.4 Evaluation . . . 33
4.5 Implementation . . . 34

5 Experiments **37**
5.1 Colorspace . . . 37
5.2 Rectification . . . 38
5.3 Augmentation . . . 39
5.4 Degree . . . 39
5.5 Datasets . . . 39
5.6 Cropping . . . 40

5.7 Visualization 42

6 Conclusion **44**

6.1 Future Work 45

References **46**

List of Figures

1.1 Example image from the dataset . . . 2
3.1 Feedforward neural network with two hidden layers . . . 12
3.2 Neural network expressed as matrix multiplications and activation functions 12
3.3 Update rules for Adam . . . 14
3.4 The sigmoid and ReLU activation functions . . . 15
3.5 The AlexNet convolutional neural network . . . 17
3.6 Dropout . . . 19
3.7 Schematic view of the VisualBackProp algorithm . . . 20
4.1 Example image from the dataset . . . 23
4.2 Two examples for roads found in the dataset . . . 24
4.3 Steering wheel angle distribution . . . 25
4.4 Example image of a busy driving scene . . . 26
4.5 Rectification . . . 27
4.6 Cropping . . . 28
4.7 Downsampling . . . 29
4.8 Mean images for two datasets . . . 31
4.9 The PilotNet convolutional neural network . . . 32
4.10 Steering wheel angle plot for trip from Berlin-Steglitz to BER . . . 35
5.1 Plot of models with different color formats . . . 38
5.2 Plot of different croppings . . . 41
5.3 Plot of different croppings with grayscale images . . . 42
5.4 Visualization of learned features for highway driving . . . 43
5.5 Visualization of learned features . . . 43

List of Tables

5.1 Evaluation of models with different color formats . . . 37
5.2 Evaluation of a rectification model . . . 38
5.3 Evaluation of a model without augmentation . . . 39
5.4 Evaluation of a model with degrees as output . . . 39
5.5 Evaluation of models trained on four different datasets . . . 40
5.6 Evaluation of different croppings . . . 40
5.7 Evaluation of croppings with a different dataset . . . 42

1 Introduction

Interest in autonomous driving and self-driving cars can be traced as far back as the 1980s [16]. The recent surge of deep learning, especially for computer vision tasks [21] has caused this interest to heighten again, with now many government agencies, academics, and industries heavily invested into their research [36]. There are already cars that can assume steering control for extended periods of time under certain road conditions like highways and the limitation that the driver remains alert and ready to take over at any moment[1].

Autonomous driving is the act of a vehicle driving itself, or rather of a designated system driving a vehicle without or only minimal human input. For this, the system uses the data from a multitude of different sources in and around the car to perceive the environment and then to make informed low-level driving decisions usually in the form of steering and speed control. The data can come from the vehicle's CAN bus, cameras, laser scanners, radars, GPS, LiDAR, route planners and others. The problem is very complex, especially since the driving system has to interact with many different agents like other vehicles or pedestrians, and is very hazardous, dealing with an already accident-prone domain, in which the wrong decisions can at best lead to heavy material damage and at worst cost lives. Most solutions fall into two different approaches [5]:

With the *mediated perception approach*, the driving scene is parsed completely using several separate sub-modules, that each focus on a different task like the detection of traffic signs and lights, pedestrians, other vehicles or lane markings. Combined, these are called the perception system [1] and their results make out a consistent representation of the immediate surroundings of the vehicle [5]. This representation is then used in the decision-making system [1] to control the car. The approach tries to comprehend the driving scene in its entirety using humanly comprehensible terms like bounding boxes of vehicles or line segments for lane markings and uses them to make safe and effective driving actions, allowing an easy understanding of how the system came to its decisions. The decisions are predictable and it can be seen in advance how the system might act to any given situation. However they also require careful human engineering and fine tuning, and if a scenario was not considered during development, the system might fail utterly if it ever occurred. Additionally, a lot of different sensors are necessary to create the complete representation, while camera data might suffice for a more direct approach. The representation is very high-dimensional, capturing a lot of superfluous details like the bounding boxes of every single car and pedestrian in the vicinity, where the distances to a select few might be enough to make informed driving decisions [5].

The *behavior reflex approach*, on the other hand, tries to create a direct mapping from sensory input to driving action [5]. Using deep learning to train a model with a dataset of human driving makes this approach very elegant and requires very little fine-tuning and engineering by the developers. It learns its own internal representations of the driving

[1] `https://www.tesla.com/autopilot`, last access on 21/05/2019

scene, allowing for a more streamlined and direct decision-making process. However, this also results in difficulties when trying to understand these decisions and trying to predict future actions by the vehicle. During driving, a lot of different agents have to interact with the autonomous vehicle [27]: passengers in the vehicle itself, pedestrians and cyclists in the vicinity, or drivers and passengers in other vehicles. Each of them must be able to trust and predict the actions of the self-driving car at least to the same amount as they would with a human driver. Several different solutions have been suggested to alleviate the black-box nature of the behavior reflex approach ([5], [2], [17]), but work still needs to be done to let this trust to be built and to allow a more complete understanding of the decision-making process. The quality of the driving system rises and falls with the quality and size of the training dataset since the model uses it to generalize to unseen scenarios. If the dataset is not comprehensive enough, the model might struggle in certain situations as a result. Whether this is the case is not always easy to tell, not at all verifiable and requires extensive testing. Additionally, the actions learned by the system are very low level like the turning of a steering wheel. Higher level decisions like lane changes are made up of several of these without actually revealing their nature. The system might not be able to grasp their concepts and when to employ them, complicating the task further [5]. On the other hand, the approach requires fewer sensor data than the mediated perception approach, as the vehicle's CAN bus is in addition to cameras oftentimes sufficient, and uses the data more efficiently. Instead of having to solve several different vision tasks in the form of the sub-modules, the behavior reflex approach regards autonomous driving as a single problem that can thus be solved more straightforwardly. While still a complex problem, the actual solution finding process and fine-tuning is undertaken by the learning system, often resulting in better performances.

The aim of this work is to train an autonomous driving model end-to-end using a dataset of human driving collected by the research project AutoNOMOS Labs[2], and it falls therefore under the behavior reflex approach. Figure 1.1 shows an example image from the dataset recorded by the front camera.

Figure 1.1: An example from the dataset recorded by a front view camera.

[2] `http://autonomos.inf.fu-berlin.de/`, last access on 21/05/2019

When designing and training a neural network for this task, many different and important decisions have to be made concerning its architecture, first and foremost of which is the shape of its input and output. What data should be used to infer the driving decisions and in what form should these come? A common choice is images of the car's surroundings and especially its front view, but the vehicle's kinematics or data from route planners as mentioned above can also be helpful. The output choices are mainly restricted to steering and speed control. While both are certainly necessary for a real driving system, a multi-task network adds in complexity and makes the training more difficult. Should past data and actions be considered? Speed control makes this almost a necessity because the speed can hardly be chosen just from a single snapshot of the driving scene without further information, for example about past traffic signs. This can be achieved by feeding a video of a few frames instead of only a single image to the network, or alternatively by using a recurrent neural network, most commonly in the form of an LSTM. Other methods can be used to influence the internal representations of the network, for example, a scene segmentation side task, or by pretraining parts of the network on other datasets like ImageNet[3]. Even after making these rather high-level decisions, many more remain, like the concrete architecture of each module or the form of the network's objective function. Section 2 gives an overview of other works training autonomous driving models using different configurations and decisions.

In this thesis, a relatively simple network called PilotNet by [3] is adopted to predict steering wheel angles from images of a front view camera. This constitutes a reduction of the normal action space of both speed and steering control, and of the more evolved network architectures developed since (see section 2), especially regarding the utilization of temporal information (e.g. in [8], [33] or [6]). However, it allows a more detailed view on possible restrictions and considerations when training a self-driving car.

Another issue is the evaluation of the trained model. Since almost every work uses a different dataset and since on-road tests are seldom feasible, the results are rarely comparable. Therefore, own baselines are used in every paper to evaluate their models, and it is difficult to estimate how these would translate to real-world driving. This problem is compounded by the fact that the recorded human behavior used for testing does not represent absolute solutions to the driving scenes. There is no one single steering wheel angle that is the perfect choice at a given time, and if the driving system predicts a deviation from the ground truth, it does not necessarily signify a wrong prediction.

To help with evaluation and additionally to help with understanding how the model learned, the VisualBackProp algorithm [2] is used to visualize regions of images that were of particular importance for the driving prediction.

[3] `http://www.image-net.org/`, last access on 21/05/2019

2 Related Work

For a long time, most works in the field of autonomous driving have been following the mediated perception approach (see section 1)[5], maybe following the general trend of focussing on different machine learning techniques other than neural networks and deep learning [12]. Still, one of the first attempts of training a driving system in an end-to-end manner was made as early as in 1989 [28] with a, by today's standards, very simple neural network with a single hidden layer. The network took a very small grayscale image together with data from a laser range finder as input and predicted classification scores for the turn curvature the vehicle would need to follow towards the center of the road. Since driving datasets of sufficient size and diversity were difficult to obtain back then, the system was trained using a small set of 1200 simulated road and range finder images. It was then tested on a 400-meter path in a wooded area using a real-world vehicle, where it seemingly performed well at low speeds, matching contemporary traditional driving systems restricted to these conditions.

Though only limited to very simple driving scenarios with few obstacles, these first experiments suggested the potential of end-to-end learning for driving [33], eliminating the need for careful and time-consuming fine-tuning by human hand required for mediated perception approaches. Once a working generator for the road images had been found, the training managed to produce a model moderately successful at following the road within only about half an hour.

More than 15 years later, [22] used a convolutional neural network to train a 50 cm long model truck to avoid obstacles on off-road terrain. It mapped two stereo YUV color-images to two real values, one for turning right and one for turning left, and was composed of five convolutional layers and a single fully-connected output layer. The training dataset consisted of about 127,000 image pairs collected in different parks and with different weather conditions, where a human expert would drive the truck at a speed of 2 m/s straight ahead until meeting an obstacle, upon which it would be steered either to the left or to the right. The system was evaluated on unseen data where a prediction was considered correctly classified when it matched the action of the human expert in terms of going left, right or straight. The results suggested that the system managed to detect obstacles and predict appropriate steering commands.

The success of deep learning in the last several years, especially since [20] won the ImageNet challenge[4] in 2012 using a deep convolutional neural network, has also led to its increased use in autonomous driving. Advancements in hardware like the highly parallelized GPUs, availability of large datasets and the development of new training techniques allow the training of deeper models which can achieve better results than traditional machine learning approaches [21]. This meant a resurgence of the behavior reflex approach for self-driving cars, but also of the use of deep learning in individual sub-modules of the mediated perception approach.

[4] `http://image-net.org/challenges/LSVRC/`, last access on 15/05/2019

[14] explored end-to-end learning for the prediction of bounding-boxes of vehicles and of line-segment representations of lane markings, that can then be further used in a complete driving system together with other modules. They collected a dataset of real-world driving on highways and used radar and LiDAR sensors to annotate the front-view images with bounding boxes and lane markings, which was then used to train a system to predict these. Their results suggest that end-to-end learning may be a very successful tool for parts of the mediated perception approach, and for autonomous driving in general.

[5] offered a compromise between the two major approaches. Instead of either parsing the scene completely using several sub-components or predicting the driving actions directly from sensor data, the model learned key perception indicators that were then used in a simple controller to generate driving commands. They called this approach a *direct perception approach* and the perception indicators functioned as a high-level representation of the driving scene, thus allowing human interpretation and alleviating the black-box problem inherent with end-to-end solutions. They included the vehicle's heading angle relative to the road and the distances to nearby lane markings and cars and were learned by a convolutional neural network based on AlexNet [20] as a mapping from front-camera images to several values normalized to the range from 0.1 to 0.9. These perception indicators were then used as input to a simple rule-based system making the driving decisions. It knew two modes based on these specific values: an in-lane mode, where the vehicle is situated firmly between two lane markings, and an on-marking mode, where the vehicle is transitioning from one lane to another. These two modes formed the basis for the driving decision. The model was trained and tested with the driving simulator TORCS[5] using a dataset of 12 hours of human driving with a focus on multi-lane highways. The model managed to navigate the game without collisions, produced better driving behaviors compared to a similar model trained using a behavior reflex approach and exceeds lane detection modules developed for the mediated perception approach. Results from testing the TORCS-trained system on images recorded with a smartphone camera during real-world driving implied that the model, though trained on virtual data, still showed promise for real-world scenarios and worked especially well for lane perception.

The system still suffered from traditional problems of mediated perception approaches, like a very limited understanding of the scene, while conventional end-to-end approaches learn a much more streamlined and optimal representation without human engineering. Thus, in 2016, [3] was the first attempt to train a self-driving car wholly end-to-end, based on [28] and [22]. The paper proposed a relatively simple convolutional neural network, termed PilotNet, that mapped the images from a single front-facing camera to steering commands. To make the model independent from the vehicle geometry, the inverse of the turning radius was used instead of the steering wheel angle. The training dataset consisted of about 72 hours of real-world driving data collected on a wide variety of roads in the USA during diverse lighting and weather conditions. In addition to the images from the front camera, it had been augmented with images generated from cameras positioned to

[5] `http://torcs.sourceforge.net/`, last access on 15/05/2019

the left and right of the original one. These augmentations represented shifts from the center of the road and their corresponding steering commands have been adjusted to lead the car back on track. They are meant to teach the model how to recover from mistakes and prevent the car from drifting off the road. The dataset had been pruned to remove its inherent bias towards driving straight and to remove images where the driver was not staying in any lane.

After training, the model was evaluated in a simulation, where the network was recursively fed an image of a driving scene that had been generated from an existing image and the previous steering prediction. In each step, the distance to the center of the lane, here called the ground truth, was computed and used to measure the model's performance: each time the distance to the ground truth exceeded one meter, a "human intervention" would trigger, resulting in 6 seconds, where the car was unable to drive autonomously, and after which the simulation was reset to the center of the lane. The actual performance of the model was then quantified by the percentage of time it was able to drive autonomously. For real-world on-road tests, a similar metric was used, where the actual time the vehicle performed autonomous steering, excluding lane changes and turns, was measured. Thus, the authors reported autonomous driving for up to 98 percent of the time for certain routes.

In order to explain how their trained model worked and made decisions, they designed the VisualBackProp algorithm [2], which created a mask of intensity values for each pixel in a specific input image indicating its influence on the prediction. This mask was then be used to visualize which regions of a scene were of particular importance for the driving task. Their results of the algorithm showed that the model has learned to recognize lane markings and boundaries as well as other vehicles and road agents [4].

Since then, several works more or less following the behavior reflex approach have been attempted. They make use of different tools in their models like scene segmentation as auxiliary tasks [6], [33], pretraining [6], [33], recurrent networks to make use of temporal information [6], [8], [33] or the learning of intermediate representations [25], use different input data like additional cameras [13], data from route planners [13], [25], vehicle kinematics like speed and vehicle heading [6], [8], or high-level commands [9] to predict different car controls like steering, speed or a general vehicle motion model [33].

[25] adopted the approach of [12] and incorporated it into a complete end-to-end system. It predicted similar visual affordances like distances to pedestrians, intersections and other vehicles, and combined them with action primitives including turning, stopping, speeding up or doing nothing. They were then used, together with the original network input, to predict the driving controls of throttle, brake and steering wheel. The network is, therefore, an end-to-end mapping of front-view images, vehicle speed and planner data to driving commands and leans towards the behavior reflex approach while retaining the original paper's benefits of human interpretability. Training and testing in the driving simulator CARLA[6] revealed significant performance increases by the usage of the auxiliary

[6] `http://carla.org/`, last access on 16/05/2019

tasks of action primitives and visual affordances compared to the same system without these. The idea behind them is to create an explicit hierarchical task decomposition, that, so the authors argue, humans perform naturally while driving.

Another work with a focus on auxiliary tasks to streamline the original driving task was [6]. It combined several different tools like semantic segmentation, optical flow, LSTMs, CNNs and transfer learning into a single neural network to predict a continuous steering angle. The authors claimed to be able to train on a relatively small dataset, while still obtaining good results. The network took raw RGB-images together with, optionally, vehicle kinematic information like acceleration, speed, heading, previous steering angle and distance to the road curb. Both the segmentation network and the CNN, acting as a visual encoder, were pretrained on much larger datasets to integrate human prior knowledge into the network. The first was trained to separate sky, road, lane markings, buildings, traffic lights, pedestrians, trees, pavement, and vehicles, while the latter was pretrained on ImageNet for basic object recognition. To incorporate temporal information, the Network used optical flow, which is a vision task that evaluates the differences between two consecutive images, and an LSTM. The input images were first processed by the segmentation and optical flow networks, whose results were forwarded to the CNN together with the original images. The resulting feature cube was then fed to the LSTM with the optional vehicle kinematics. In the end, a fully-connected network made the steering angle predictions. The model was trained and evaluated using both the comma.ai dataset[7] of around 7 hours of real-world driving and a dataset of virtual driving. The performance of the network was measured in the number of failures, where in the simulation a lane boundary violation and in the real-world ten consecutive prediction errors with a magnitude of more than 5 degrees constituted as failures. Examination of the model using different configurations of auxiliary tasks and network architectures revealed that each module improved the performance against a baseline without any auxiliary tasks, while the segmentation task contributed the most.

The previous paper used only a very small dataset of actual driving data and compensated by pretraining both the segmentation and the convolutional networks. [33] on the other hand relied on a very big, crowd-sourced dataset of about 10.000 hours of driving. It has been recorded using a multitude of different vehicles and at different locations using dash-cams and smartphone sensors, which makes it highly diverse, but also, lacking accurate vehicle sensors for speed and steering wheel angle, very imprecise. They used scene segmentation as a privileged learning side task, but, in contrast to the previous paper, only used its loss in the training task, thus influencing the internal representation learned by the network. They further designed the model without specific driving commands as output in mind, but rather to learn a generic vehicle motion model, that can then be adopted as needed. The complete network was termed FCN-LSTM using an altered AlexNet [20], pretrained on ImageNet, as a visual encoder and an LSTM as a temporal encoder. To evaluate the system, it has been trained both using a discrete

[7] `https://github.com/commaai/research`, last access on 16/05/2019

action and a continuous driving model as the vehicle motion model. The first consisted of four actions: going straight, stopping or making a left or right turn. The paper reported accurate predictions of human actions, like stopping at traffic lights or when a preceding car braked. The latter predicted classification scores for the angular speed of the vehicle, distributed into 180 bins. With this, the side task improved the performance of the model significantly compared to the base network, especially when reacting to traffic lights.

[8]'s focus lay in the utilization of temporal cues between consecutive images. For this, its network took a stack of 15 images as input, which were fed into a "feature-extracting sub-network" consisting mainly of spatiotemporal-convolutional, or st-convolutional, layers. These are able to consider temporal dynamics between images on top of the spatial dynamics within a single image that normal convolutional layers are capable of. The predictions were made by the "steering-predicting sub-network" that used the extracted features together with further vehicle states like previous vehicle speed, torque, and wheel angle to predict the next driving commands for speed, torque, and wheel angle using an LSTM module. The model has been trained and tested using the udacity real-world driving dataset[8] augmented by a flipped version of each data point. The paper explored different methods of data reduction like cropping the top-region of the images, sub-sampling to lower spatial or temporal resolutions or keeping only the driving scenes with high steering angles. Each method produced adverse results, with the exception of spatial subsampling that had no effect either way. The system has been further analyzed with an adaption of the VisualBackProp algorithm [2], indicating that the model managed to train key visual indicators like lane markings or other vehicles.

[13], [24] and [7] put their focus on finding bettersuited input data for the driving task. Each built on either a more comprehensive set of sensors in the first case or more specialized sensors in the latter cases.

[13] introduced a dataset consisting of 60 hours of human driving, recorded using, besides vehicle sensors for steering wheel angle and vehicle speed, 8 cameras delivering a 360 degree surround view and two different kinds of route planner data: a video of the rendered map of the planned route in TomTom Maps and a stack of GPS coordinates for the past driving trajectories together with the GPS tags for the next 300 meters using OpenStreetMaps. They argued that constraining the model input to front-view and vehicle kinematics limits its necessary knowledge of the driving scene and hinders its ability to make informed driving decisions. Humans would use a more complete view of their surroundings using rear and side mirrors, as well as a mental map of the path to their goal. They examined the benefits of the dataset by training a network using one of the two route planners together with image data from four of the eight cameras to predict both vehicle speed and steering angle. The data from the TomTom route planner improved the prediction of the steering angle significantly, while the data from OpenStreetMaps had no clear improvements. This seemed to stem from the fact that TomTom already provides structured information in the form of a rendered map, compared to the GPS coordinates

[8] `https://github.com/udacity/self-driving-car/tree/master/datasets`, last access on 16/05/2019

by OpenStreetMaps, which additionally suffer from GPS inaccuracies. The surround-view cameras were especially useful for speed-control at intersections and in low-speed city scenarios where the view of other cars is of importance.

[24] used Event-Cameras, cameras that instead of recording images at set intervals, report intensity changes in separate pixel-specific events. These were then used to create event-frames by accumulating them in a specific pixel over a set time interval. They trained a convolutional neural network using event-frames to predict steering angles and compared the results to networks trained with either conventional grayscale images or the difference between two consecutive grayscale images. Their test results suggest that the event-frames outperform grayscale and grayscale-difference images especially for shallower CNNs and especially at high speeds. They argued that event-frames are better able to cope with abrupt lighting changes. They reported performance improvements compared to other state-of-the-art models, due to the ability of event-cameras to capture scene dynamics. Conventional grayscale images would suffer from blur at high speeds, while event-frames preserve edge details. On the flip-side, event-cameras seemed to perform worse at low speed since relatively few pixel-intensity changes are reported.

And finally, [7] learned a driving policy from LiDAR point-clouds in addition to conventional camera data. They argued that the depth information they provide offer crucial information for the driving task to the model. Their test-results suggest that LiDAR sensor data can greatly improve the driving models compared to attempts with only camera and video data.

[34] tried to alleviate the problem of domain shifts between data collected from different sources. They proposed finding a virtual representation for each image in a real-world dataset using Generative Adversarial Networks in combination with a dataset of virtual driving scenes recorded in TORCS or CARLA. The network learned to generate the virtual representations at the same time as a steering angle predictor like PilotNet [3]. This forces the generator to generate images that still retain necessary semantics and yield the best prediction performances. These representations would then have further advantages like a reduction of noise and the removal of irrelevant scene components like shadows and would also take advantage of virtual driving data and simulators that otherwise would not be useable for models designated for real-world applications. The Results of their tests suggest success over conventional end-to-end learning approaches like [3].

None of the works discussed so far allow human control during inference without disengaging the driving system altogether. [9] suggested a model that allows high-level human commands in the form of different behaviors at upcoming intersections at runtime. They argue that imitation learning makes the implicit assumption that the driving decisions are fully explained by the input to the model, but that this would not be the case for high-level decisions like lane changes or turns at intersections when limiting the input to sensor data. Therefore, they augmented datasets of both driving with a 1/5th scale model truck and urban driving in CARLA with a set of commands supposed to indicate decisions at intersections. They then designed and trained a conditional neural network

with a different fully-connected network depending on the current command. Since the set of high-level commands is rather small, the number of different branches is limited. During testing in CARLA, the system could complete a predefined route with a set goal more often than not. In the physical system, the truck managed to complete the set course competently with minimal human intervention and no missed intersections, implying that the proposed model is a promising attempt at introducing high-level commands into autonomous driving systems.

[17] incorporated the detection of salient image regions as seen in [2] into the actual prediction task of the network. For this, their network consisted of three parts: first a visual encoder in the form of a CNN, then a "coarse-grained-decoder" which used a combination of soft attention mechanisms and an LSTM-unit to predict the steering wheel angle and produce a heat map of attention and as the last step a "fine-grained decoder" which applied a causality test to the heat map regions as to whether they causally influence the steering prediction by masking them out of the image and observing for drops in performance. Test results showed that the incorporation of the attention did not result in worse performance of the model, which paves the way for models that do not only predict the steering angle in a black-box manner but at the same time provide attention maps that allow reconstruction of the decision making process.

3 Deep Learning

Deep learning is a technique in the field of machine learning, in which the system learns a hierarchy of representations from presented experience and data to model complex concepts. Each representation forms a layer of abstraction defined by their relation to the previous one [12]. Thus the system might be able to understand a driving scene in terms of lane markings and vehicles, by first learning to detect edges, then motifs as an arrangement of edges, then objects formed from motifs and at last compositions of objects [21].

Most notably neural networks (see section 3.1) can be used to approximate functions in the form of complex input-output relations by discovering intricate structures of patterns in high-dimensional data [21]. They are trained with supervised learning (see section 3.2) together with gradient descent (see section 3.2.1) and the back-propagation algorithm (see section 3.2.2) to predict output to a given input, like steering wheel angles given driving scenes in the form of the vehicle's front view.

3.1 Neural Networks

A *neural network* is a collection of units that are connected with each other by weighted edges. They use an activation function to process the sum of their inputs and send the result through their outgoing edges [26]. Inspired by the human brain, they're often called neurons.

A *feedforward neural network* is an acyclic neural network, in which information flows exclusively from input to output without feedback connections [12]. Its neurons are usually arranged into separate layers where each neuron may only receive input from the previous layer and send activations to the next. The first one is called *input layer* and receives the input from outside the network, while the last one is called *output layer* and produces the output of the network. Their number of units specify how many values or features can be fed to the network and how many predictions are output by the network [29]. Between input and output layer lies an arbitrary number of layers, each with an arbitrary number of units, called *hidden layers* since they are not visible from outside the network [19]. Figure 3.1 shows a schematic view of a feedforward neural network with two hidden layers.

Instead of a network of nodes and edges, this structure can also be interpreted as a nested function of matrix multiplications and activation functions (see Figure 3.2). Each layer receives a vector of input values that is multiplied by weight matrices, fed to the activation function of the layer and then sent to the next layer [12]. This concept is usually used to implement neural networks, making use of heavily parallelized vector and matrix computations.

Neural networks are able to model a wide variety of complex functions, denoted by the *capacity* of the network. The capacity of a given neural network is mainly dependent on the number of hidden layers in the network, the number of units in each layer, the connections between units, and the choice of activation function for each layer. The specific function

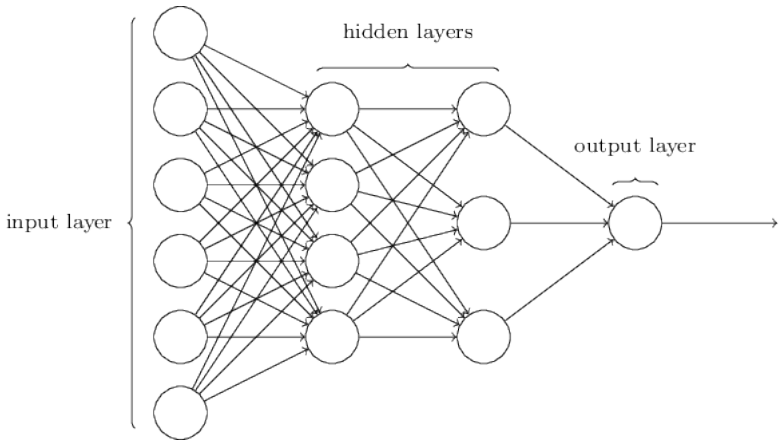


Figure 3.1: A schematic view of a feedforward neural network with two hidden layers [26].

$$f(x) = \sigma_2(W_2\sigma_1(W_1x))$$

Figure 3.2: Neural network with a single hidden layer expressed as a nested function of vector matrix multiplications and activation functions with weight matrices W_1, W_2 and activation functions σ_1, σ_2.

expressed by the network is then defined by the values of its weight matrices, also called the parameters of the network. The act of finding good configurations for these parameters, which is finding a function that solves the designated task, is called *training* a neural network. [12]

A *fully-connected (feedforward) neural network* is a feedforward neural network, where each neuron in a layer is connected to each neuron in the subsequent layer [23].

3.2 Training

Training a neural network is an iterative process during which it learns from a set of training samples. A sample consists of the representation of a single problem together with its corresponding solution or label, either in the form of the actual answer or in the form of a subjective answer given by a human expert. In each iteration, the predictions produced by the network are compared to the corresponding labels using a loss function that calculates an error reflecting the current performance of the model. The error is then used to adjust the parameters of the network in a way that minimizes the error. [12]

The complete function that computes the error given the network weights, input and desired output is called *objective function* and looks as follows:

$$J(\Theta) = \frac{1}{N}\sum_{i}^{N} L(f(x_i;\Theta), y_i) \tag{3.1}$$

with network parameters Θ, network function f, loss function L and N training samples $\{x_i, y_i\}$. Thus, training the network basically means solving the optimization problem

defined by the objective function with regards to the weights. The best solution would then correspond to its global minimum, which is almost always unattainable. Therefore the training is usually stopped when it reaches a performance plateau with no further significant improvements. [12]

This method of training a neural network using a label as reference is called *supervised learning* and relies heavily on the size and diversity of the training dataset. As a result, the collection of adequate samples is of particular importance. Since the actual objective of the model is not its performance during training, but rather during inference, it is evaluated using a second dataset, called validation set, that has been collected separately. The results of this evaluation are then used to decide whether the network performs well enough for its designated task or whether it needs further training, a larger training dataset or whether its design must be adjusted. [12]

3.2.1 Gradient Descent

The adjustment of the network's weights based on its error is performed using an algorithm called *gradient descent*:

$$\Theta = \Theta - \alpha \cdot \nabla_{\Theta} J(\Theta) \tag{3.2}$$

Given the output of the objective function J, its gradients are calculated with regards to the weights Θ, which are then updated by a step in the negative direction of the gradients. This is the direction of the steepest descent of the objective function with regards to the weights, thus most likely leading towards an acceptable solution. The size of the step α is a hyperparameter called *learning rate*. [23]

In order to update the parameters with the actual average gradient of the objective function, the network would need to be evaluated on the complete training dataset each time. Since complex problems necessitate very large datasets and deep networks, this would mean that training would take a very long time and be basically impractical. Instead, a variant of gradient descent called *mini-batch gradient descent* or *stochastic gradient descent* is used, with which only a relatively small batch of about 50-250 samples of the training set is used for a single update. The calculated gradient is then only a noisy estimate of the real average gradient but is in practice good enough. Each update step using a mini-batch is called one *iteration* and one iteration over every mini-batch of the whole dataset is one *epoch*. [30]

Stochastic gradient descent (SGD) solves the problem of the required time for a single update of normal gradient descent, but still has some key problems like the dependence on a good learning rate that can make or break the performance of the network [30]. To remedy some of these difficulties, several variations of SGD have been developed, like SGD with momentum, which uses a velocity depending on past gradients to update the parameters. This is designed to help if the gradient gets caught between two slopes of the objective function but when the actual minimum lies along the ravine. With SGD, the gradient

would make only small progress towards the minimum, but rather oscillate between the two slopes. The momentum term introduces speed along the common direction of past gradients and corrects the update step towards the minimum. [30]

The most common variant of SGD is *Adam* [18], which is based on a number of different variations. [30] describes it as, instead of a ball rolling down a slope like SGD with momentum, a heavy ball with friction, thus coming to rest on local minima on the surface of the objective function. Adam keeps two moving averages: an estimate of the first moment of the gradient, the mean, and the second moment of the gradient, the variance. Besides its behavior on the objective function, it has the added benefit of decaying the learning rate as the training progresses. [18]

$$g_t = \nabla_\Theta J(\Theta_{t-1}) \tag{3.3}$$
$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \tag{3.4}$$
$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \tag{3.5}$$
$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{3.6}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{3.7}$$
$$\Theta_t = \Theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{3.8}$$

Figure 3.3: The update rule of the Adam variation of stochastic gradient descent. [18]

3.2.2 Backpropagation

Every variant of gradient descent still needs the gradient of the network with respect to its weights. For this, *backpropagation* is used. It is based on the chain rule, that states how the derivative of two nested functions is calculated, and the fact that the network is already composed of many small units, each with a near enough differentiable activation function. The algorithm then simply consists of traversing the network backward, starting with the output of the loss function and ending at its input layer. In each unit, the local gradient with regards to its input is calculated and multiplied with the gradient from its output as instructed by the chain rule. The resulting gradient is then used to calculate the gradient with respect to the weights and send to the next unit as its output gradient. [12]

This constitutes as an efficient and easy to implement algorithm to calculate the gradients for each iteration. It is one of the key reasons why deep learning is so successful, and its versatility means it can not only be used for simple feedforward networks but also more complex layers like convolutional layers (section 3.5), batch-normalization layers (section 3.7) and recurrent networks. Even some activation functions can be learned using the backpropagation algorithm [12].

3.3 Activation Functions

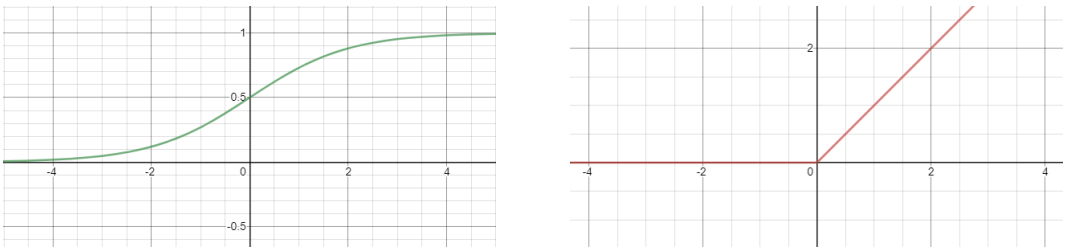


Figure 3.4: The sigmoid (left) and ReLU (right) activation functions.

A neural network with none or exclusively linear activation functions would only ever be able to model linear functions itself, limiting its capacity and its ability to learn the desired task severely [12]. The weight matrices would collapse into a single matrix and the network would then perform only a simple matrix multiplication on the inputs. Instead, each neuron usually applies a non-linear activation function to the sum of its incoming values and emits the result.

Activation functions take a real value and produce another real value. They are differentiable in almost all points as a prerequisite for the backpropagation algorithm (section 3.2.2) used for training the network. There are several different activation functions, each with its own advantages and disadvantages. For a long time, the sigmoid function (see Figure 3.4 left)

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.9}$$

has been the usual choice, because it squashes its inputs into the open interval from 0 to 1, modeling the idea of a neuron not firing at all for small, and firing at its maximum frequency of 1 at large inputs [23]. Unfortunately, this makes the unit only truly sensitive at values around 0 and saturate towards 0 or 1 everywhere else, meaning that the gradient is almost 0 at these points, which effectively kills the gradients and discourages learning [12].

The most common choice nowadays is the *rectified linear unit* (see Figure 3.4 right)

$$ReLU(x) = \max(0, x), \tag{3.10}$$

because it works well on most problems. It causes a subset of all units to be inactive for a given input and performs linearly on the rest, retaining the advantages of linear activation functions without limiting the network capacity. Some of these advantages are ease of computation, good flow of gradients, short training times and an easy understanding of how it works in conjunction with other functions. It forces actual sparsity onto the network by making a number of units inactive leading to several advantages (see [10] for more details).

On the flipside, the number of units necessary to model complex functions increases

significantly with ReLU's compared to other activation functions. Since the gradient is 0 if the unit is inactive and 1 otherwise, the gradients do not vanish, but the weights of the unit might instead be updated in a way that causes the output to always be 0, leading to dead units that can not be restored due to a gradient of zero. [23]

The fact that ReLU's are not differentiable in 0 is not an issue since an input of 0 is very unlikely and subject to numerical error anyway. Implementations usually make the decision to either return a gradient of 0 or 1 at this point. [12].

There are variants of the ReLU that address some of these problems, like the leaky ReLU, that instead of returning 0 for values smaller than 0 returns a small scale of the actual value, attempting to alleviate the dead unit problem. In most cases, and especially with careful weight initialization or Batch-Normalization (section 3.7), the normal ReLU is a good and sufficient choice. [23]

3.4 Loss Function

The *loss function* calculates the error of a set of predictions compared to the corresponding labels, and thus mainly depends on the type of output of the network. If the network is designed to produce a set of classification scores, each score indicating how likely the presented input belongs to a certain class, the loss function has to reflect this and aim to increase the score of the correct class while reducing the scores for the rest. [12]

The prediction of a continuous, real-valued steering wheel angle is a regression problem, and the error must reflect how well or how badly the network performed compared to the human expert, and where a loss of 0 indicates a perfect prediction. The most common choice for problems like these, where a quantity with a specific real-world interpretation is predicted, is the *mean squared error* (MSE), which simply computes the squared L2-distance between predictions and labels and averages the result over the number of predictions:

$$L(x, y) = \|x - y\|_2^2 . \tag{3.11}$$

It has a simple derivative and produces an error that can easily be interpreted in the context of the problem. [26]

3.5 Convolutional Neural Network

A *convolutional layer* is a specific type of neural network layer that allows the detection of patterns anywhere in the input [21]. This is particularly useful for images, in which neighboring pixels form composite features like the nose of a person or the bumper of a car regardless of their position in the image. The relative spatial relations between features are important, rather than the absolutes.

This is realized with weight-sharing, with which there is no specific weight for every single input value, but rather a small filter that is used on every part of the input. Since the input to a convolutional layer is usually an image, the filters can be thought of as

matrices, that are slid along the spatial dimensions of the image, performing the dot-product at each location along the way. The resulting *feature map* is then processed by the activation function and sent to the next layer. [12]

Instead of only applying a single filter to the input, convolutional layers usually consist of several filters stacked on top of each other, each with separate parameters and each learning a different pattern. Thus, the layer does not produce a single feature map, but rather a whole *feature cube* [17]. To deal with input in the form of these or images with several channels, for example red, green and blue, filters stretch along the whole depth of the input dimensions [23]. Figure 3.5 shows a schematic view of the AlexNet convolutional neural network architecture taken from the original paper [20]. It shows the 224x224x3 input image and the resulting feature cube of each convolutional layer.

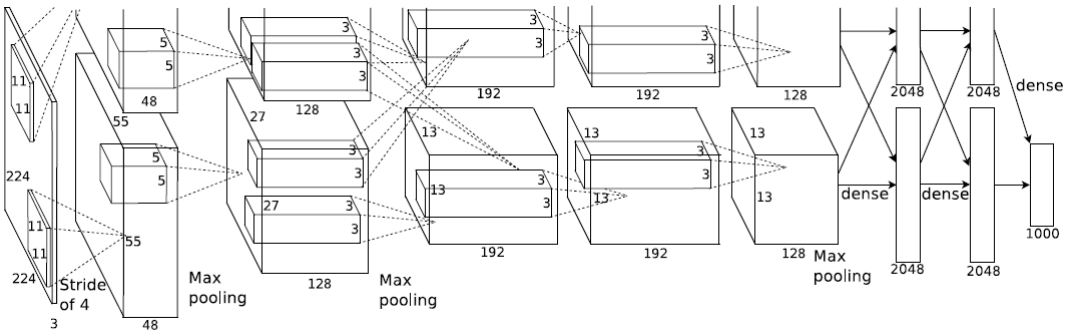


Figure 3.5: A schematic view of the AlexNet convolutional neural network [20].

Hyperparameters of convolutional layers are not their number of units like conventional layers, but rather the size and count of their filters, and additionally their stride and padding. Instead of performing the convolution at every feature, it might only be performed at every second or third position, effectively downsampling the input. Depending on the size of the filter, it can not be applied to the edges of the image, requiring padding of the input with zeros or accepting further downsampling. [12]

A *convolutional neural network* usually consists of several consecutive convolutional layers, each learning subsequently higher levels of abstractions. With an image as input, earlier layers might detect several different edges at every position in the input, which are then used to learn motifs, then parts, objects and finally composites of objects in later layers [21]. Thus, a stack of convolutional layers might learn to detect vehicles, lane markings or pedestrians in a driving scene. A few fully-connected layers can then use the flattened final feature cube to make driving decisions. The convolutional layers are understood to perform 'feature extraction', bringing the input into a representation more suitable for learning, while the fully-connected layers make the actual decisions [21]. Therefore, a convolutional neural network is sometimes called an image encoder [33].

A *deconvolutional layer* performs the inverse of a convolutional layer, effectively upsampling the input. They are commonly used for image segmentation or similar applications in encoder-decoder networks, in which several convolutional layers encode the input image

into a higher level representation, which is then decoded again using deconvolutional layers to recreate a variation of the original image with differently colored pixels for specific objects. [35]

3.6 Regularization

Training a neural network with supervised learning has the inherent drawback that the network learns to perform well on the training dataset, while its actual purpose is a good performance in every scenario and not just those present during training. The network has the tendency to *overfit*, memorizing instead of learning, and generalizing badly to unseen data. Techniques that try to alleviate this problem are known as *regularization*. They do not concern themselves with improving the training error, might in fact worsen it, but rather with the generalization error [12].

One of the easiest ways to achieve this is by extending the training dataset and especially improving its diversity. A driving system that has only been trained on data recorded during the day, will have trouble performing well at night time. If collecting new data is not possible or too expensive, augmenting the dataset with samples generated from existing ones might go along way towards improving generalization. Common augmentation techniques include adding noise, mirroring or rotating the data points and masking regions of the input. Which augmentation strategies work well differs from case to case. Further, a moderately complex function has no trouble fitting a very small number of data points perfectly, but will then perform very badly on other points from the same distribution. Forcing the function to compromise between a huge number of points will lead to worse performance on these points, but it will have a higher chance to perform well on unseen data. [12]

The same effect can be achieved by reducing the complexity of the function. In neural network terms, this would mean limiting the network's capacity by reducing its number of layers or the number of units in the layers.

The same can be done by using a practice called *weight decay*, with which a regularization term is appended to the loss function and thus restricts the parameter values. Most commonly, the L2-Norm of the weight matrices is used for this, penalizing big matrices and driving its values towards the origin [12]. The objective function then looks something like this:

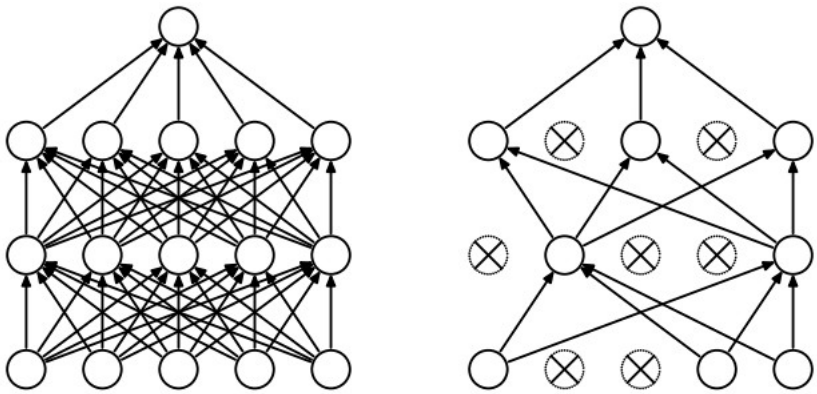
$$J(\Theta) = \frac{1}{N} \sum_{i} L(f(x_i; \Theta), y_i) + \lambda \sum_{w \in \Theta} w^2 \tag{3.12}$$

where Θ are the parameters of the network and λ is, as a hyperparameter, the weight of the regularization term.

3.6.1 Dropout

A good way to improve generalization in machine learning is the use of ensembles, in which multiple models are trained independently either using different datasets or different techniques. The final prediction of the ensemble is then the weighted average over all models. If done correctly, this can result in a significant increase in performance, since every model puts importance on different features. Unfortunately, this is not feasible for neural networks, since even the training of a single network often requires a large dataset, a lot of computational power and a lot of time.

Dropout [31] is a variation of this idea. For every training iteration, every neuron has a chance of p to be temporarily removed from the network. This can be seen as, instead of training a single complete network, training a different sub-network for each iteration (see Figure 3.6). Every sub-network is only trained for a single update, but since they share the parameters with the original network, every parameter has a chance of $1 - p$ to be updated in any given iteration. During application time or testing, the complete network is then used without dropout, but in order to keep the estimated output of every neuron the same as during training, their output is multiplied by p. This is equivalent to averaging the prediction of every potential sub-network to obtain a single complete prediction that works well on unseen data. [31]



(a) Standard Neural Net (b) After applying dropout.

Figure 3.6: The complete network (left) and a specific sub-network for a single iteration, in which crossed units are dropped (right). [31]

3.7 Batch Normalization

The distributions of the inputs to the layers of a neural network change naturally during the training process when the parameters of the previous layer are updated. This forces the layers to continually adapt to these new distributions in every iteration and therefore slows training considerably, in part because it necessitates lower learning rates. The authors of [15] call this the "internal covariate shift" and they propose a new type of network layer called *batch normalization* aimed at reducing this.

The batch normalization layer uses the same principle as mini-batch gradient descent, in that it calculates the mean and variance of its inputs for a single batch and uses them to estimate the corresponding values for the complete dataset. They are then used to normalize the layer's input per feature. Combining a batch normalization layer with every normal layer in the network ensures consistent distributions of their inputs. The authors recommend inserting the batch normalization between linear transformation and activation function of a layer. During inference, the mean and variance calculated during training are used. One of the key parts of the technique is the fact, that the backpropagation algorithm works on it, enabling learning on the normalized inputs. [15]

In addition to the speedup of the training, batch normalization has been shown to reduce the need for good parameter initialization and to act as a regularizer, boosting the generalization of the network and reducing the need for dropout and L2 weight decay. [15]

3.8 VisualBackProp

The VisualBackProp algorithm [2] was designed by the developers of [3] to show how their driving model PilotNet learned how to steer a car [4]. It is used to visualize the parts of an input image that influence the network's prediction the most. Its idea is to combine the learned representation of the convolutional layers into a single mask with intensity values for each pixel corresponding to the intensity of their activations and thus to their importance for the prediction task.

The algorithm uses the feature cubes of the convolutional layers during the backward pass of training, hence the name. The feature cubes are averaged over their depth to create an average map of the layer's activations. Then, starting at the last convolutional layer in the network, each average map is deconvoluted (see section 3.5) to the size of the previous layer and multiplied with that layer's average map. This is repeated until the first convolutional layer of the network has been reached. See Figure 3.7 for a schematic view of the algorithm.

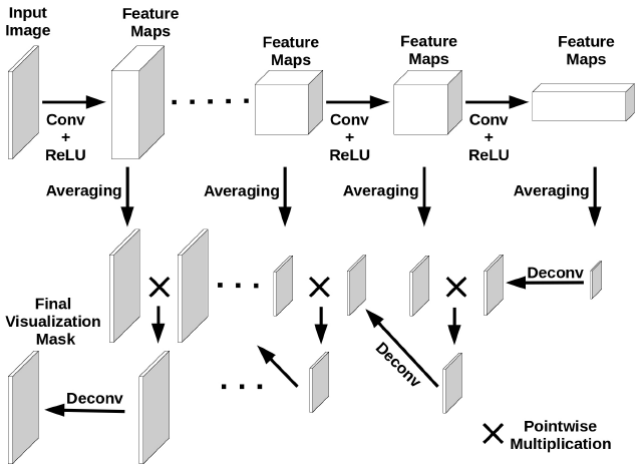


Figure 3.7: A schematic view of the VisualBackProp algorithm. [2]

The last deconvoluted map is normalized to the interval [0,1] to receive the final visualization mask with the same spatial resolution as the original input image. This mask can then be layered on top of the image to create a visualization of the pixels most crucial for the network task.

4 Approach

The aim of this work is to train a direct mapping from images to steering controls using driving data collected by the research project AutoNOMOS Labs[9] and belongs therefore to the class of behavior reflex approaches for autonomous driving (see section 1).

The mapping is realized by a convolutional neural network based on PilotNet [3] and uses images from a single front facing camera mounted in front of the windshield to predict steering wheel angles. Thus, the model can be understood to make driving decisions based on the front view a human driver would have while driving the car.

As mentioned, this is a reduction compared to the real world, where the driver has control over both steering and speed. He additionally has a more complete view of the surroundings using mirrors, a mental map (or even a real one) of the planned route and a memory of past events, like his driving decisions and the positions of other agents on the road. Newer network architectures (see section 2) take these and others into account to create a more complete and better performing driving system, but the simplification of PilotNet allows a more detailed view on the basics of training a self-driving car, considering the scope of this work. Therefore, in this and the following chapter (section 5), some basic decisions for training data, data format, network architecture and model evaluation are explored and compared.

4.1 Dataset

4.1.1 Collection

The entire dataset has been recorded over the course of more than two years during different weather and lighting conditions on roads in and around Berlin in Germany. The vehicle used was a Volkswagen Passat equipped with several cameras, laser scanners, GPS, radars and LiDAR. Data from these sensors and the vehicle's CAN bus was recorded, processed and archived using ROS[10] while a human expert drove the car. The result is a set of .bag-files, each containing from a few seconds to an hour worth of driving data, where every sensor measurement is coupled with its own timestamp.

40 bag-files were chosen for this work based on their length, route and available sensor data, equaling about 13 hours of driving on urban roads and highways. This amounts to a relatively small dataset compared to the 72 hours of the paper [3] this work's network architecture is based on. This will result in worse performance on unseen data, as the model's generalization improves with the size of the dataset, but should still suffice to gain an idea of the process needed to train a driving system. Other works ([6]) have been successful in training more evolved models using small datasets with the help of pretraining both the convolutional layers and a scene segmentation side task, while again others ([33]) used vastly larger datasets of 10.000 hours of crowd-sourced but imprecise driving data to train their models (see section 2 for more details).

[9] `http://autonomos.inf.fu-berlin.de/`, last access on 21/05/2019
[10] `https://www.ros.org/`, last access on 19/05/2019

For this work, the images of the front view and the steering wheel angles were extracted with ROS and paired based on their timestamps with a tolerance of 0.1 seconds. The images were then saved in separate folders for each bag-file as jpegs together with index files matching steering wheel angles to images. The images were sampled at a rate of 5 frames per second, because higher rates would result in many very similar images, bloating the dataset without adding any real value and hindering training. In the end, the complete dataset for this work consists of about 230,000 image - steering wheel angle pairs. See Figure 4.1 for an example image from the dataset.

Figure 4.1: Example image recorded with the front-view camera.

4.1.2 Filtering

For the model to be able to learn steering wheel angles based on image data, there needs to be a clear correlation between angles and images [9]. If their magnitude increases, the reason must be apparent in the input provided to the network, for example by the curving of the road. However, this is not always the case: the expert's decision to change the lane or make a turn at the next intersection might be completely arbitrary or dependent on the trip's destination. Without additional information like data from a route planner or high-level commands indicating these decisions, the model will have trouble finding a connection between driving scene and steering, resulting in predictions little better than a random guess. Adding further input or using a more evolved network architecture might be able to deal with these issues (section 2), but that is beyond the scope of this work. Instead, they have been filtered out of the dataset, as has been done in [3]. For this, a tool has been written to allow an expert to view the dataset and control whether the shown image - steering angle pairs should be written to new index files to later be used for training or not.

As a further simplification of the driving task, two more sub-datasets have been created. For the first, all roads have been filtered out, where their lane boundaries were not

immediately apparent for at least one side. See the left image in Figure 4.2 for an example in which no lane markings exist and in which the lane boundaries are made up of parking cars. Removing samples like these allow a greater focus on visible lane markings and boundaries, and should simplify learning steering control for following the road. The second dataset takes this one step further, as it is limited to highway driving since highways usually have well-maintained road surfaces and lane markings and a more predictable and orderly traffic flow [14]. The right image in Figure 4.2 shows an example image from the highway dataset. After filtering, the three datasets consist of about 150,000, 110,000 and 73,000 images respectively. With a sampling rate of 5 fps this roughly translates to 8, 6 and 4 hours of driving data.

Figure 4.2: Example images from the dataset, in which either the road boundaries are very unclear because of parking cars and missing lane markings (left) or in which the roads are very well-maintained with clear lane markings as seen on highways (right).

4.1.3 Pruning

The vast majority of roads are for the most part straight interspersed by relatively short curves. With the added fact that driving at high speeds requires relatively little steering, the result is, that the dataset has a prevalence of low steering wheel angles, especially after removing lane changes and intersections. See Figure 4.3 for the distribution of steering wheel angles on the complete dataset (left). Training the model on such a dataset without consideration would naturally lead to an autonomous vehicle with a preference for driving straight. The car would leave the road at the first moment steering became necessary. There are a few possible solutions for this problem:

- Collecting more data and seeking out roads with as many curves as possible.
- Removing the majority of all driving samples with low steering wheel angles.
- Pruning the dataset as a pre-processing step for training by randomly sampling for different road curvatures and restricting the number of straight road images ([3]).
- Upsampling images with greater steering wheel angles by adding copies to the training dataset ([6]).

- Applying weights to the loss-function, with which images with larger steering wheel angles weigh more ([33]).

In this work, the third method is used. For this, every sample in the training dataset is sorted into one of 40 different bags depending on their steering wheel angle. The actual set for training is then created by sampling a pre-determined maximum number of data points from each bag, thus limiting the number of training images for each steering wheel angle and especially for driving straight. The cutoff number for each bag is a hyperparameter for training and must be selected based on the distribution and size of the specific dataset. Removing too many images will result in a too small dataset and therefore a model that generalizes badly to unseen data, but not removing enough will limit the effect of the pruning step. Figure 4.3 (right) shows the pruned dataset.

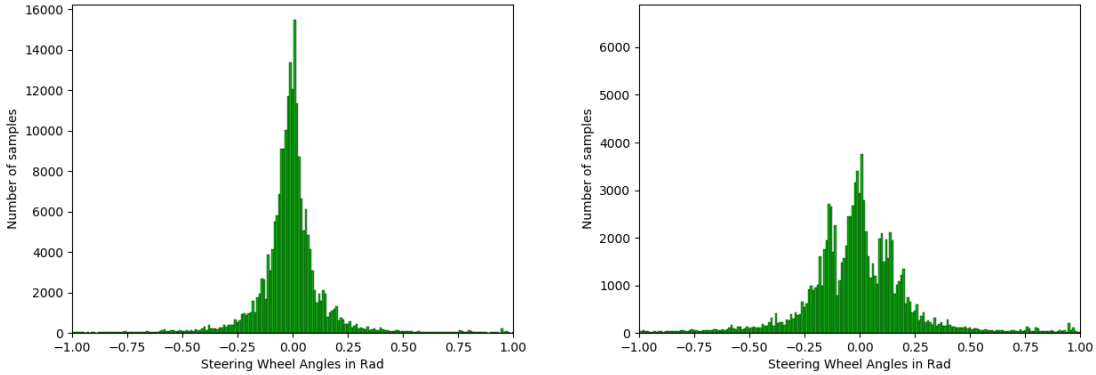


Figure 4.3: Steering wheel angle distribution on the complete dataset before (left) and after pruning (right).

4.1.4 Augmentation

Augmenting the dataset with additional images generated from existing ones is a common step to improve the model's ability to generalize (see section 3.6). [8] added horizontally flipped copies of each image and [6] additionally varied their brightness randomly. [9] randomly changed contrast, brightness and tone, and applied noise and region dropout. [3] generated horizontally shifted versions of each image with the use of two additional cameras that were mounted to the left and right of the original, front view camera. The aim was to simulate the vehicle drifting off the road and supplying adjusted steering wheel angles that steered it back towards the center. This would allow the system to learn to recover from mistakes.

Here, only horizontal mirroring is used to augment the dataset. [8] warns, that this can lead to the model learning violations against the traffic rules, by flipping right of ways and turning right turns into left turns, as well as mirroring text on traffic signs. However, these disadvantages require a level of complexity beyond the scope of this work. The hope is to

reinforce the correlation between steering and road curves and to remove an imbalance of turns to either side.

4.2 Preprocessing

One of the questions that have to be answered while designing a neural network is which form both its input and output should take. While the general idea of mapping the driver's front view to the steering controls has already been decided by the choice of network architecture, their actual shape leaves some room for exploration. The following preprocessing steps are possible transformations on the original data, each with advantages and disadvantages. Which configuration of transformations work best with this specific dataset and network will be examined further in section 5.

The images were recorded using a fisheye camera with a resolution of 1280x800 in the RGB color format. The nature of the camera leads to visual distortions resulting in a higher field of view with objects directly in front of the camera appearing closer [32]. This should be especially useful for the driving task since most of the crucial information resides in the middle and to the sides of the view. Figure 4.4 shows a driving scene with a busy roundabout. The top and bottom parts of the image hold no relevant information, while the center shows the road with the cars ahead, the lane markings and the vehicles in neighboring lanes, and might show traffic lights and signs in different scenes. However, the corners show nothing at all due to the circular nature of the image. Since the number of pixels that can be used for training is limited because of computation and storage constraints, this would constraint the number of relevant features further and limit the model's capabilites.

Figure 4.4: Example image of a busy roundabout taken with a fisheye camera.

4.2.1 Rectification

One way to solve this problem would be by rectifying the fisheye images to obtain regular, rectangular ones, with the downside of reducing their field of view at the same time. See Figure 4.5 for the rectified version of the earlier image of the roundabout. The corners now show part of the view, but the blue car in the neighboring lane that was about to overtake the motorcyclist is no longer visible. On top of that the image looks unfocused and especially the pixels in the corners look distorted. It is likely that this does not constitute as an improvement over the circular images. The neural network can simply disregard the pixels in the corners and while it looses some of its input space this way, the information content of the input is still higher.

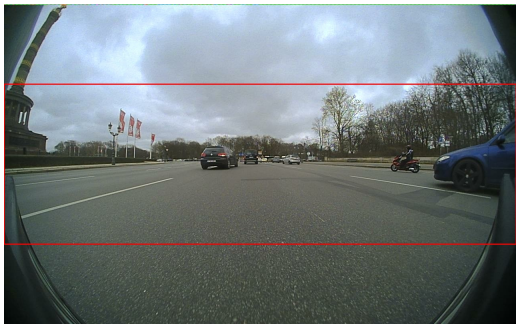
Figure 4.5: Rectified version of the image in Figure 4.4

4.2.2 Cropping

As mentioned, several parts of the image have no real importance for the driving task. The corners show nothing due to the circular nature of the images, the bottom shows the immediate ground in front of the car, and the top shows mostly the sky. Removing these parts by cropping the images can reduce their noise, streamline the training by focusing the task on the truly relevant details and increase the information content of the input pixels. During earlier iterations of the development of the network architecture and the training process, the model at one point focused heavily on the sun visible in the sky and used it almost exclusively for its driving decisions. While the model was terribly overfitting, memorizing the steering wheel angle as per the position of the sun and the current iterations are in no real danger of this, the point still stands: Noise and extraneous elements can influence the training, even if they would never have a true influence on the decisions a human driver would make.

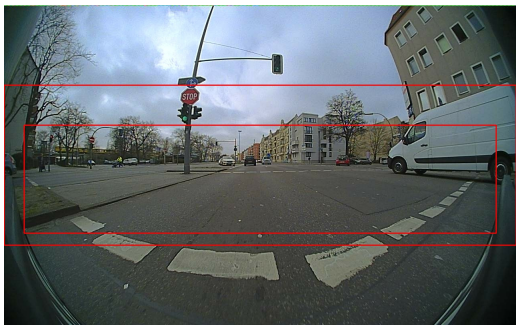
The question then remains of how much to cut. Figure 4.6 on the next page shows

several possible bounding boxes, marked in red, for the new image cutouts. The cropping in Figure 4.6a looks like a safe choice with everything of importance still included, but with remaining parts of the sky and the casing of the camera still visible. Figure 4.6b shows a much more focused cropping, cutting parts of the car at the right side, but minimizing the amount of irrelevant details. In other scenes, like shown in Figure 4.6c, this might cut some rather more important features than the back parts of a car. The first cropping would here at least still include the lower of the two traffic lights together with the signs, but with the second one, they are cut off completely. The amount of cropping depends therefore mainly on the ambitions of the driving system. If it is expected to detect traffic signs and lights and take them into consideration for the driving task, the cutout must be chosen carefully and tested on many different driving scenes to ensure that all important details are included. If the aim is rather to mainly follow the road and maybe react to other cars, a more modest cutout might suffice. Figure 4.6d takes this to the extreme, retaining almost exclusively the lane markings. This might allow a model trained only on highway data to perform nicely at following the road, but would limit the driving task a lot, to the point where the system might hardly be called a driving system or a self-driving car.



(a) A possible, but rather large cropping of the earlier roundabout scene.

(b) A smaller and more focused cropping of the roundabout scene, but cutting off parts of the car to the side.



(c) The croppings from (a) and (b) applied to a different scene.

(d) A very small cropping focused on lane markings.

Figure 4.6: Possible croppings of different driving scenes.

Since the images will be downsampled before training (section 4.2.3), the cropping does not change the size of the network input.

4.2.3 Downsampling

The resolution or size of the input to the network depends on a few different factors. If chosen too high, the images will take up more storage space, require longer loading and decoding times, and take more space in memory during training. While convolutional layers (section 3.5) do not really care about the resolution of their input since the filters slide over their spatial dimensions regardless of their size, if chosen too small the forms depicted by neighboring pixels might become unrecognizable. If the same objects are made up of a greater number of pixels, either bigger filters to detect these or deeper networks and thus a higher number of layers of abstraction become necessary. This may especially play a role for cropped images, since, if the input resolution is unchanged, the same amount of pixels are available to depict less of the original image. Filters or a succession of filters that before were able to detect an object in its entirety may now struggle to do so.

Figure 4.7 shows the earlier scene with (right) and without (left) cropping after being downsampled to the resolution of 200x66 used by PilotNet [3]. Some of the important features like the left lane markings or the motorcyclist as well as some of the preceding cars almost become unrecognizable without cropping. While this may still be manageable for convolutional layers, the same filters might not be able to detect the bigger versions of the same objects. It is also of note, that the aspect ratio of the images is not retained, putting again the focus on the horizontal dimension of the image compared to the vertical one.

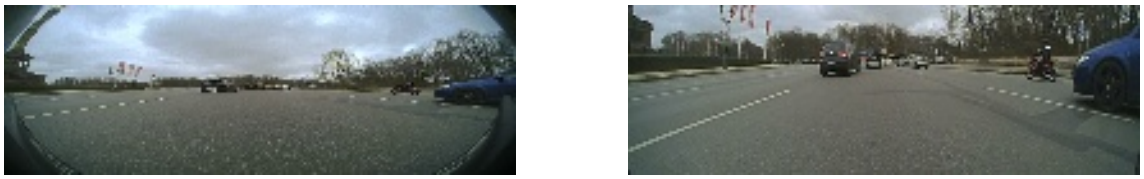


Figure 4.7: Downsampled versions of the earlier scene without (left) and with (right) cropping.

The images are resized using bilinear interpolation.

4.2.4 Colorspace

Most transformations between color spaces are linear transformations and thus in theory learnable by the network itself, but the choice of color format might still play a small role, especially if grayscale images are considered. [3] used the YUV color space, [6] and [17] used HSV and [8] used RGB. So there seems no clear leaning to either of them.

RGB, HSV and YUV each store three values for each pixel. RGB stores red, green and blue intensity values, which should help with the differentation between different colored objects, but might hinder the detection of edges and contours. HSV and YUV have a separate value for the brightness of each pixel, which should be helpful for edge detection, but HSV reserves only a single value for the color which might limit the model on that

front, since a relatively small change of the value already brings a rather big change in hue. In the end, the choice between these three is not expected to matter much.

Grayscale images only hold a single intensity value for each pixel, forgoing color completely. This is, therefore, a simplification of the driving task again, reducing the noise in the images at the expense of potentially important information like the color of signs and traffic lights, lane markings, brake lights, and vehicles.

4.2.5 Normalization

Normalizing the input features is a common step for computer vision tasks and machine learning in general. Before training, the mean and standard deviation of the entire training dataset is computed per pixel and per channel as follows:

$$m_{ijk} = \frac{1}{N}\sum_{n}^{N}(x_n)_{ijk} \tag{4.1}$$

$$\sigma_{ijk} = \sqrt{\frac{1}{N}\sum_{n}^{N}((x_n)_{ijk} - m_{ijk})^2} \tag{4.2}$$

where N is the number of samples in the training dataset, $(x_n)_{ijk}$ is the k-th channel of the ij-th pixel of the n-th image in the training dataset. During training and inference, each image is then subtracted by the mean and divided by the standard deviation before being fed to the network:

$$\hat{x}_{ijk} = \frac{x_{ijk} - m_{ijk}}{\sigma_{ijk}} \tag{4.3}$$

This ensures that the input distribution of every feature is centered around zero and has a variance of one, allowing for faster and smoother training.

Since a mean value for every channel in every pixel is calculated, a mean-image of the complete dataset can be constructed. See Figure 4.8 for both the mean images of the complete dataset and of the highway dataset. The horizon, sky and road are clearly visible and for the complete dataset, the right road boundary is vaguely perceptible, which is probably due to the fact, that the dataset was recorded while driving on the right side of the road. In the image for the highway dataset, the lane markings and road boundaries on both sides are very well defined, which enforces the earlier statements (see section 4.1.2) of how a restriction to highway driving makes the task much easier.

4.2.6 Steering Wheel Angles

The steering wheel angles are stored as radians and lie between -10 and 10. As mentioned before (section 3.4), the prediction of a single continuous value is a regression problem, which brings some limitations, for example on the use of loss function. Instead, the values could be separated into a finite, discrete set of bags and solved as a classification problem as has been done by [28] or [33]. The network would then predict scores or possibilites

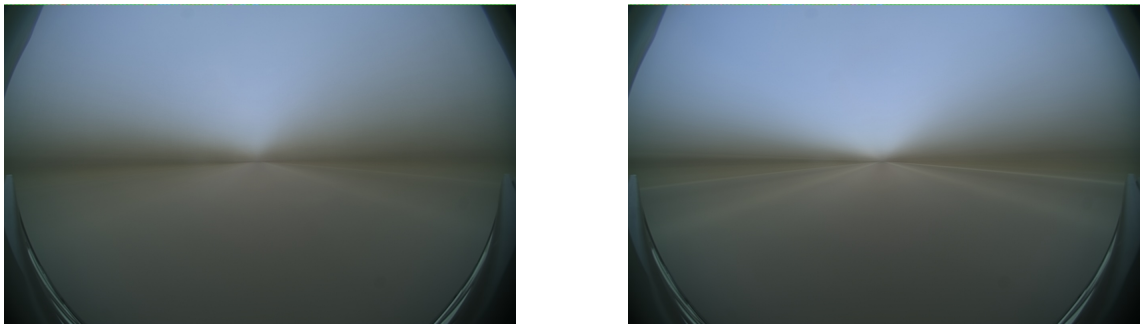


Figure 4.8: The mean images of the complete data (left) and of the highway dataset (right).

for each bag indicating how likely the corresponding steering wheel angles were used by the human expert. Of course, this does limit the accuracy and possibly the smoothness of consecutive predictions, and while it has advantages for the training process [23], this experiment has not been attempted in this work.

[3] transforms the steering wheel angles into the inverse of the turning radius of the vehicle, to make the model independent from the car geometry, since the same steering wheel angle can result in different turns depending on the vehicle architecture. The inverse is used to avoid a singularity when driving straight. This again is beyond the scope of this work, since an independence from the vehicle used to record the dataset is not yet of concern for this model.

Another idea, that indeed will be tried for this work, is the transformation of the steering wheel angles from radians to degrees, but the benefit of this is dubious, since this transform is, similar to the colorspaces (section 4.2.4), a linear one and the network should not care either way about the form of the angles.

4.3 Network Architecture

PilotNet [3] is a neural network of five convolutional layers and three fully-connected layers. Its input are 200x66x3 images in the YUV colorformat, that are normalized to have zero mean and unit standard deviation before being fed to the first convolutional layer. The first three convolutional layers have a stride of 2x2 and a filter size of 5x5 and the last two have a stride of 1x1 and a filter size of 3x3. Their inputs are not padded, therefore with each subsequent layer the size of the feature map decreases further. The feature cube of the last layer is flattened into a single vector and then passed to three fully-connected layers with 100, 50 and 10 neurons. All eight layers use the ReLU activation function (see section 3.3) [2]. A last single neuron without activation function connected to every neuron of the last fully-connected layer makes the final prediction in radians. See Figure 4.9 for a schematic view of the network. The network is trained using the mean squared error loss function (see section 3.4) and consists of about 250 thousand parameters.

The network architecture is a very simple one, especially compared to today's state of the art networks with several million parameters, skip connections or additional modules

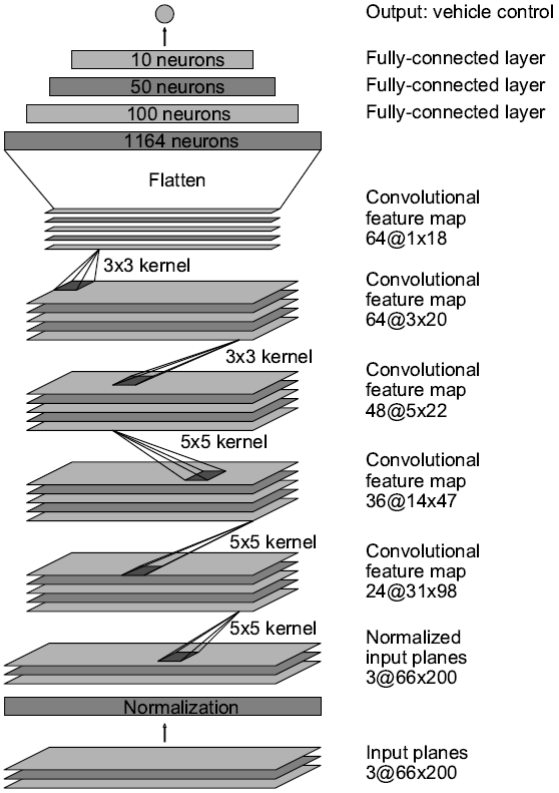


Figure 4.9: The PilotNet architecture designed to predict the inverse of the turning radius from the front view of a driving scene. [3]

like LSTMs. Curiously the network does not rely on pooling layers to reduce the spatial size of the input as is common in convolutional networks [23], but rather on convolutional layers with a stride greater than one and by forgoing padding the input with zeros. AlexNet [20], the winner of 2012's ImageNet challenge[11], has a similar structure with five convolutional layers and three fully-connected layers, but uses a much larger filter size in the first layer and makes heavy use of pooling layers (see Figure 3.5 in section 3.5).

The paper that proposed PilotNet [3] does not make further comments about the network architecture and training details, therefore it is unclear whether the following elements are present in the original model or not. The addition of batch normalization layers (see section 3.7) between every convolutional and fully-connected layer and their activation function improved the performance of the network immensely, speeding up the training and ensuring better generalization to the validation dataset. To improve upon this further, two more methods of regularization (section 3.6) were added: A L2 weight decay term was appended to the objective function and dropout used for every network layer with a drop probability of 50%. These measures decrease the performance on the training set, reducing overfitting and lead to better results on the validation set.

The weight matrices are initialized according to the Glorot normal initialization [11], though the choice of initialization seems to not have made much of a difference for training,

[11] `http://image-net.org/challenges/LSVRC/`, last access on 15/05/2019

likely because of the batch normalization layers. Adam was used for training the network (see section 3.2.1), with default values for the hyperparameters β_1 and β_2.

4.4 Evaluation

Evaluating a driving model can be difficult, since field tests are oftentimes not feasible due to risks to human life and of material damages. [3] was able to use their driving system to follow the road on certain routes and reported an autonomy of the self-driving car for 98% of the time, excluding turns and lane changes. One alternative to real-life testing, is the use of simulations like TORCS[12] and CARLA[13], but this in turn necessitates the training of the model on virtual data, which throws the actual value of the testing into question. [3] designed a simulator with the use of their existing driving dataset. The images are manipulated based on the predicted steering and the distance to the center of the lane calculated. A human intervention would then trigger each time the distance exceeded 1 meter, resulting in a resetting of the simulation to the center of the lane and an autonomy penalty of 6 seconds. The success of the model was then measured in percentage of the time it was able to drive autonomously in the simulation (see section 2). Such a simulation is unable to measure anything else than the system's ability to follow the road, and is very expensive and time consuming to design and calibrate.

Lacking any of these, metrics and visualizations for the evaluation on a validation or test dataset have to be found, that express the performance of the model in a meaningful way. Even then, the comparison to other works is difficult since the use of different datasets throws off any parallels that can be drawn. Oftentimes, the papers can only measure against models they have trained on their own datasets, like simple baselines of their own architectures or relatively simple networks, where the choice often falls to [3]'s PilotNet.

A common choice is then to simply use the loss of the network, which is usually the mean squared error (see Equation 3.11). Since this gives the error compared to the labels of the dataset in terms of squared radians, the root mean squared error (RMSE) or alternatively, the mean absolute error (MAE) can be used:

$$rmse(X) = \sqrt{\frac{1}{N} \sum_{x,y \in X} \|f(x) - y\|_2^2} \tag{4.4}$$

$$mae(X) = \frac{1}{N} \sum_{x,y \in X} |f(x) - y| \tag{4.5}$$

where X is the test dataset, N its number of samples and f the network function. However, these metrics do not express how big a single error can get. The vehicle might be able to drive perfectly for most of the time, but if it leaves the lane or the road in between, this might not be reflected properly with these metrics.

[12] `http://torcs.sourceforge.net/`, last access on 15/05/2019
[13] `http://carla.org/`, last access on 16/05/2019

Another issue lies in the labels that have been recorded while a human expert drove the car. There is not one definite way to drive, and there is no single optimal steering wheel angle to a given driving scene. While controls by the human expert are a viable choice and actual mistakes are rare and should be caught while filtering the dataset, a network prediction that differs from the corresponding label might be applicable just as well. When evaluating on an existing test set using human decisions as ground truth, this problem can only hardly be circumvented. A possibility is an accuracy metric as seen in [7]. A prediction is classified as correct if it deviates from the label by at most 0.1 radians or about 6 degrees. The performance of the model is then measured by the percentage of correct steering wheel angles. This allows some freedom from the ground truth and might also give a better idea of the magnitude of the errors. [6] took this one step further and counted failures, where 1 second's worth of predictions each deviating from the ground truth by more than 5 degrees constituted as a failure. This metric is a bit softer and might allow for example the taking of a curve to begin a few frames earlier or later, but a failure would then be seen as way more critical than a few incorrect predictions as per the accuracy metric.

[6] proposed one more metric to measure the smoothness of the predictions: the mean continuity error (MCE):

$$mce(X) = \sqrt{\frac{1}{N-1} \sum_{i=1}^{N-1} (f(x_{i+1}) - f(x_i))^2} \tag{4.6}$$

where N is again the number of samples in the dataset and x_{i+1} and x_i two consecutive images from the dataset. A high value would indicate big jumps between predictions, which is of course undesirable.

A way to actually visualize both the predictions and the ground truth is to plot them for a single trip over time. Figure 4.10 is the plot for a highway drive from Berlin-Steglitz to the airport BER after removing lane changes.

The Visualbackprop algorithm (see section 3.8) can be used to further investigate the model. While it does not really give an indication about the quality of its predictions, it can give a clue of how it came to make these predictions. The algorithm can be used to mark those pixels of an image, that had the biggest impact on the network output. With a video or single images showing what the model regarded as important in the driving scene, it can be verified, that it actually learned something instead of simply memorizing the dataset or making random guesses dependent on noise in the images. The expectation is that it at least learned to recognize lane markings and boundaries and uses them to predict steering wheel angles in order to follow the road.

4.5 Implementation

The code for this thesis is made publicly available[14].

[14] `https://git.imp.fu-berlin.de/meyerjan/autonomous-driving`, last access on 22/05/2019

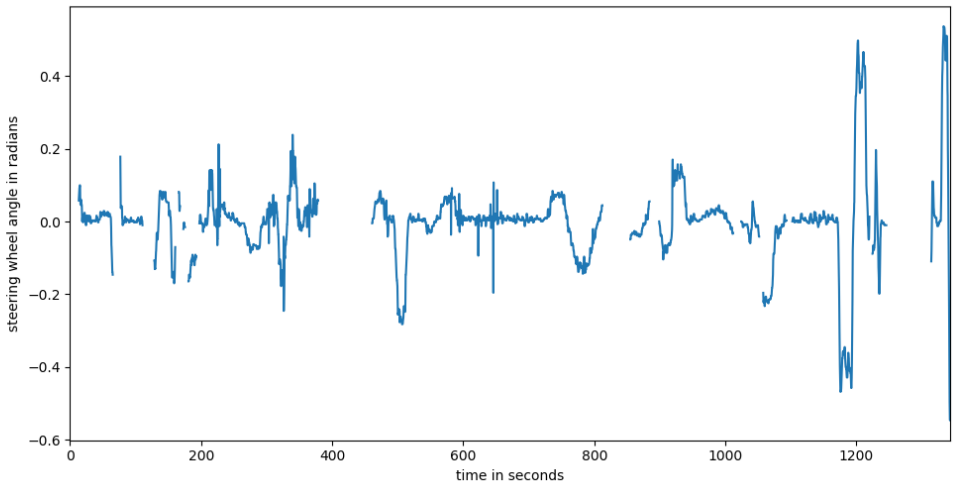


Figure 4.10: The steering wheel angle in radians over time for a highway trip from Berlin-Steglitz to BER after removing lane changes.

As already mentioned, the extraction of the training samples is performed using ROS. The pre-processing steps, training itself and evaluation are implemented in python using tensorflow[15], matplotlib[16], open-cv[17] and numpy[18]. The filtering is performed using a program written for this thesis in C# with WPF[19].

The images and steering wheel angles are extracted from the .bag-files, potentially rectified (see section 4.2.1), synchronized and saved in a separate folder for each .bag, in which the images are stored as .jpgs with their timestamp as name and matched to the corresponding steering wheel angles in a text file called 'index.txt'. During the filtering (see section 4.1.2) additionally text files are created with only those image - angle pairs that are to remain in that particular dataset. As preparation for training, a python script is used to gather all samples from one of the datasets, divide it into a training and a validation set, prune the training dataset according to section 4.1.3, calculate its mean and standard deviation (section 4.2.5), augment it (section 4.1.4), perform the preprocessing steps (section 4.2) as needed and at last store them into separate .tfrecord-files. For the validation set a few trips are randomly selected, amounting to about 80% of the available samples. No images from the same .bag-file will appear in both training and validation sets at the same time, to ensure the validity of the generalization error calculated during validation. The tfrecord file format is a format designed specifically for tensorflow and is part of its tf.data library to enable faster loading times during training and reduce necessary storage space.

The network architecture is implemented using the tf.layers library. During training

[15] `https://www.tensorflow.org/`, last access on 21/05/2019
[16] `https://matplotlib.org/`, last access on 21/05/2019
[17] `https://opencv.org/`, last access on 21/05/2019
[18] `https://www.numpy.org/`, last access on 21/05/2019
[19] `https://github.com/dotnet/wpf`, last access on 21/05/2019

the training samples are loaded with the tf.data API from the tfrecord files and shuffled to ensure that the batches (see section 3.2.1) represent the complete dataset as best as possible. The images are parsed into tensors and then fed to the network. The tensorflow API calculates the network prediction and the resulting loss and performs the update step. After each epoch, a status updated is issued and the current performance on the validation set measured. If it exceeds the performance of previous epochs, the current network state, that is its parameters and settings, is saved. At the end of training, the best performing network state is then the trained model, instead of just the last one. Additionally, a log of the training progress together with plots for mean absolute error and loss function during training are saved.

For evaluation or visualization the network state is restored from the saved models, and used to predict steering wheel angles and generate the visualization masks as detailed in section 3.8. These can then be either used to calculate the model performance using metrics or to layer the masks on top of the original images to visualize their salient regions either in separate images or as a stream of images in a video.

5 Experiments

Several different options for network input and output were considered in section 4. Some of these will be examined further in the following chapter by training a number of models with different configurations each. Unless specified otherwise, they used the highway dataset (see section 4.1.2) each with an identical training - validation set split, to make them as comparable as possible. The training set was pruned with a cutoff of 10,000 samples per bag (see section 4.1.3) and augmented with the flipped version of each image. Training lasted for 60 epochs with an initial learning rate of $\alpha = 0.00025$, a L2 weight decay term with a weight of $\lambda = 0.001$ and a batch size of 64.

As a reference point for each configuration, a model trained using the original circular images on the highway dataset without further modifications was used. To establish this baseline, its colorspace will have to be selected first, therefore this is the first point of experimentation.

5.1 Colorspace

The four colorspaces RGB, HSV, YUV and grayscale were highlighted in section 4.2.4 with the expectation that the choice would not matter much, with the possible exception of grayscale images, since their structure is actually different from the other three color formats due to their removal of colors altogether.

Table 5.1 shows the results of the metric evaluations of a model trained with each. Their performances do not differ much, even for the grayscale images. The only notable difference seems to be the mean continuity error of RGB, indicating smoother steering compared to the others, with smaller jumps between consecutive predictions. Figure 5.1 shows the predictions of the four models against the corresponding labes for a small stretch of the Berlin-Steglitz to BER trip shown in Figure 4.10. While the plot for the RGB model is visibly smoother than the others, this does not seem to translate to a better performing driving system. None of the models manage to reliably take the four curves at timestamps 500, 575, 750 and 780.

Colorspace	MSE	RMSE	MAE	MCE
RGB	0.00786	0.08867	0.0536	**0.02824**
HSV	**0.0077**	**0.08778**	0.05785	0.04068
YUV	0.0079	0.0889	0.05581	0.03727
Grayscale	0.00784	0.08855	**0.05526**	0.0389

Table 5.1: Evaluation of four models on the validation dataset, each trained with a different colorspace, using the mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE) and mean continuity error (MCE) metrics.

Since the colorspace, as expected, does not seem to matter much, YUV is selected for the next several experiments, because this was the choice of [3] as well.

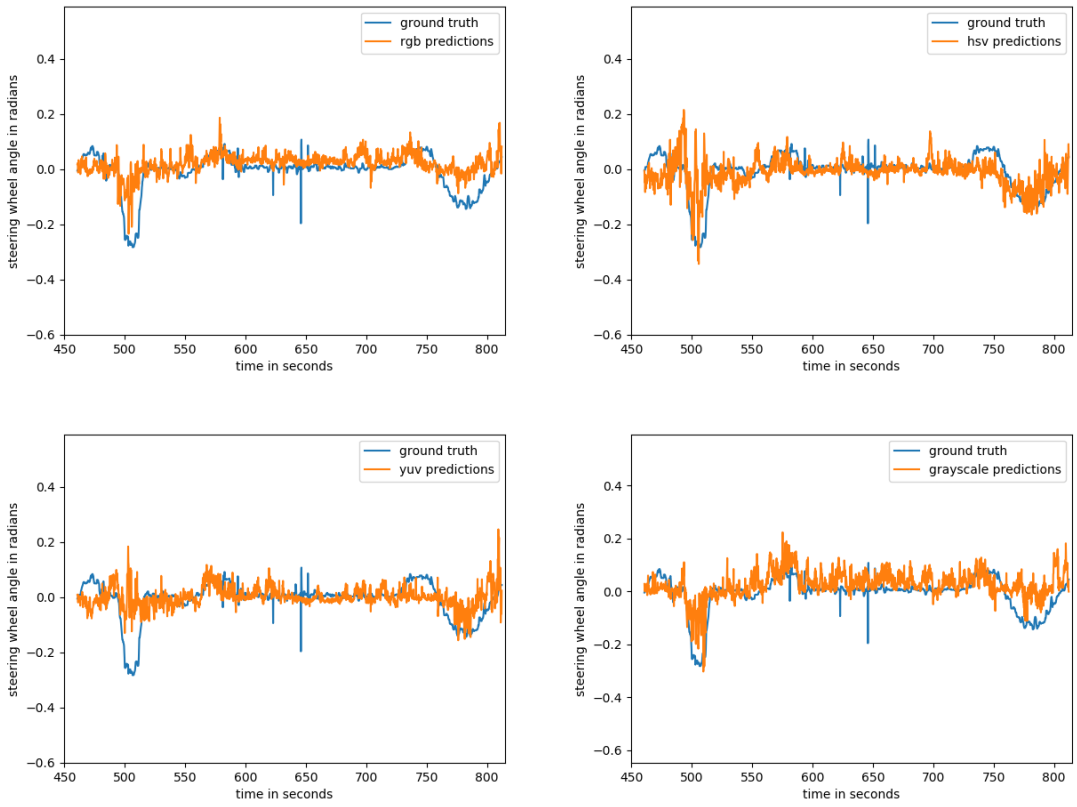


Figure 5.1: Plot of the predicted steering wheel angles for the RGB (upper left), HSV (upper right), YUV (lower left) and grayscale (lower right) models for a small stretch of the Berlin-Steglitz to BER trip, uninterrupted by lane changes.

5.2 Rectification

Configuration	MSE	RMSE	MAE	Accuracy
YUV baseline	**0.0079**	**0.0889**	**0.05581**	**0.86067**
YUV with rectification	0.0092	0.09595	0.06157	0.82826

Table 5.2: Evaluation of the YUV baseline model with and without rectification.

Rectification has been considered as an option to deal with the disadvantages coming with the fact that the images were recorded using a fisheye camera (see section 4.2.1). However, this removes their advantages at the same time. The results shown in Table 5.2 suggest, that the advantages of the fisheye camera outweigh the disadvantages. The model trained using the rectified images performs notably worse than the YUV baseline. This is especially visible when using the accuracy metric proposed in section 4.4. While the drop in performance is not catastrophic, it is obvious that this option is not really worth considering further.

Configuration	MSE	RMSE	MAE	Accuracy
YUV baseline	**0.0079**	**0.0889**	**0.05581**	**0.86067**
YUV w/o augmentation	0.00845	0.09191	0.0576	0.86053

Table 5.3: Evaluation of the YUV baseline model with and without augmentation.

5.3 Augmentation

Table 5.3 shows the baseline YUV model trained with and without augmentation in the form of mirrored versions of each training sample. While the improvements obtained with the augmentation are not significant, they are nonetheless enough to indicate its usefulness. Every step gained when training a neural network, no matter how small, is important to eke out the maximum possible performance.

5.4 Degree

Changing the nature of the output of a neural network can be very delicate. Changes to the input between two models will still allow a relatively easy comparison between the two. However, the change from radians to degrees as the output unit, changes the unit of the MSE, RMSE and MAE metrics as well. Table 5.4 shows the evaluation of the two models. Since the root mean squared error and mean absolute error are given in radians and degrees respectively, the results can be compared after converting them to one or the other.5.30905° are about 0.09266rad and 3.29242° about 0.05746rad, therefore there seems to be a surprisingly clear advantage of radians over degrees.

Configuration	MSE	RMSE	MAE	Accuracy
YUV with radians	0.0079	**0.0889**	**0.05581**	**0.86067**
YUV with degrees	28.18603	5.30905	3.29242	0.84205

Table 5.4: Evaluation of the YUV baseline model with radians as output and with degrees as output.

5.5 Datasets

As detailed in section 4.1.2, four datasets have been created in this thesis: the complete, unfiltered dataset (A), the dataset without any lane changes or intersections (B), the previous dataset without any roads, where the lane markings and boundaries were not immediately apparent (C) and the highway dataset without any lane changes (D). Table 5.5 shows the evaluation of four models trained on these datasets. The one trained on (D) is the YUV baseline. The results clearly reflect the importance of removing high-level decisions like lane changes and intersections, since the performance of the corresponding model is a lot worse than any other model. Furthermore, it is obvious how much the restriction to highway driving helped the system. During training, the network tries to find correlations between the input images and the steering wheel angles, and at the same

time as there are none for turns at intersections, it is all the more clear for the orderly and well-marked road surfaces of highways. The filtering that turned the dataset (B) into (C), that is the removal of roads with unclear boundaries, improved the system's performance some, but maybe to a lesser extent than expected.

Dataset	MSE	RMSE	MAE	MCE	Accuracy
(A)	0.94804	0.97367	0.35058	0.34012	0.46452
(B)	0.09027	0.30045	0.1128	0.05012	0.72216
(C)	0.01709	0.13072	0.07598	0.03844	0.76278
(D)	**0.0079**	**0.0889**	**0.05581**	**0.03727**	**0.86067**

Table 5.5: Evaluation of models trained on the four datasets using the YUV color space: (A): complete dataset, (B): (A) without lane changes and intersections, (C): (B) without roads where the lane boundaries are not apparent, (D): (B) restricted to highway driving.

5.6 Cropping

Three different options for cropping the input images were considered in section 4.2.2: one that was careful not to remove any crucial information, but therefore still retaining irrelevant data (1), one that made sure to remove most of the extraneous details at the expense of some relevant ones like traffic signs and lights (2), and one that removed almost everything except lane markings (3).

Table 5.6 shows the results of these croppings trained on the highway dataset in both YUV and grayscale color formats. The model performance improves greatly with each successive cutout, while cropping (3) predictably performs the best. Cropping (1) already has a rather big effect, considering that it only removed unnecessary features like the sky, that the network should be able to filter out on its own. It is clear, that cropping is a key pre-processing step, worth consideration when training any driving model.

A plot of the predictions (see Figure 5.2), again on the stretch of the Berlin-Steglitz to BER trip, indicates that the model gets progressively better at following the curves with each cropping. The model with the narrow focus on lane markings obviously performs the best, but the inclusive cropping of (2) already shows promising road following behavior, without sacrificing too many details.

Configuration	MSE	RMSE	MAE	MCE	Accuracy
YUV without Cropping	0.0079	0.0889	0.05581	**0.03727**	0.86067
YUV with (1)	0.00727	0.08526	0.05267	0.03811	0.87486
YUV with (2)	0.00587	0.07661	0.04807	0.04407	0.89878
YUV with (3)	0.00441	0.0664	0.04531	0.04267	0.91576
Grayscale with (2)	0.00438	0.06618	0.0452	0.04329	0.91298
Grayscale with (3)	**0.00421**	**0.06491**	**0.04177**	0.0397	**0.93057**

Table 5.6: Evaluation of models trained using the three different croppings proposed in Figure 4.6: (1): biggest, (2): midsized, (3): smallest.

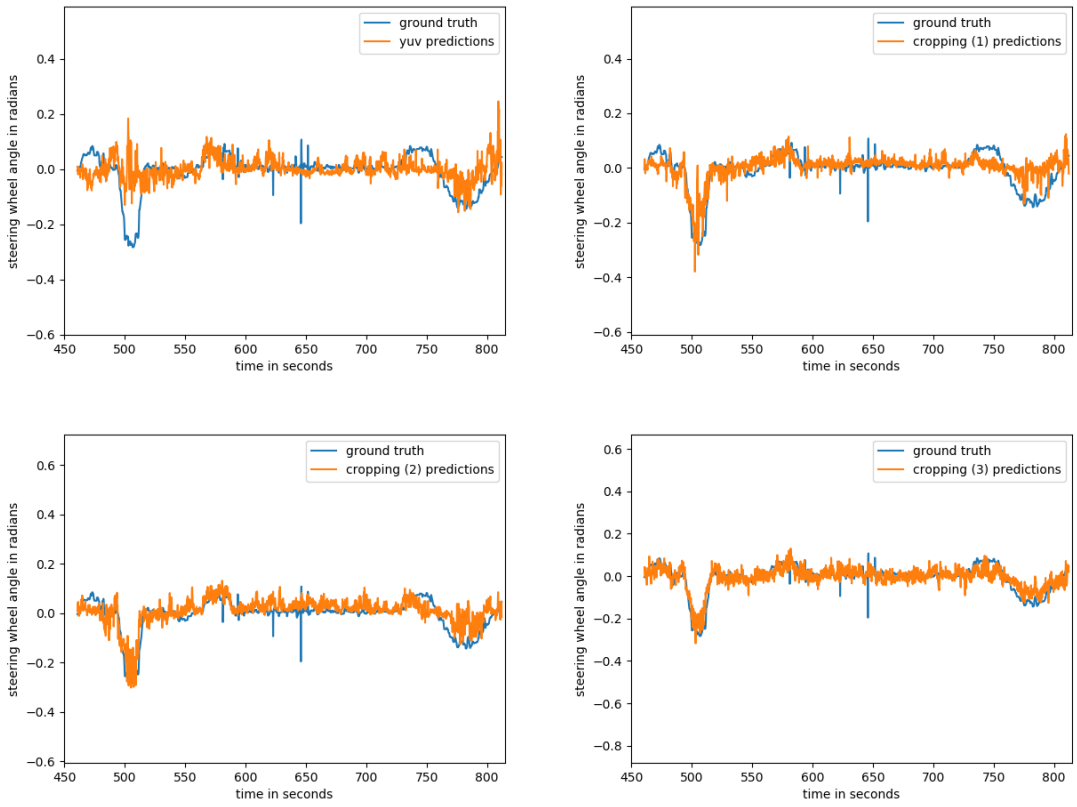


Figure 5.2: Plot of the predicted steering wheel angles for the YUV baseline (upper left), YUV with cropping (1) (upper right), with (2) (lower left) and with (3) (lower right) models for a small stretch of the Berlin-Steglitz to BER trip, uninterrupted by lane changes.

Curiously, while the usage of grayscale images did not show any noticeable difference with the earlier models, they show significant performance increases when using croppings compared to the equivalents trained on YUV images. Cropping (2) already reaches the same performance as YUV with cropping (3), and cropping (3) results in the best model in terms of metric evaluations found in this thesis. Maybe the elimination of noise and extraneous details performed by the croppings are further enhanced by the same effect of grayscale images. Figure 5.3 (top) shows the plottings of grayscale with (2) and (3). While the curves are still not taken perfectly, and consecutive predictions are not smooth, the model reacts to the curves very well.

Figure 5.3 (bottom) shows the same plot for the first big curve at timestamp 500. The models follow the steering by the human expert surprisingly well, and if their function was smoothed further to remove the jagged behavior, it is imaginable that the predictions might hit the ground truth almost perfectly.

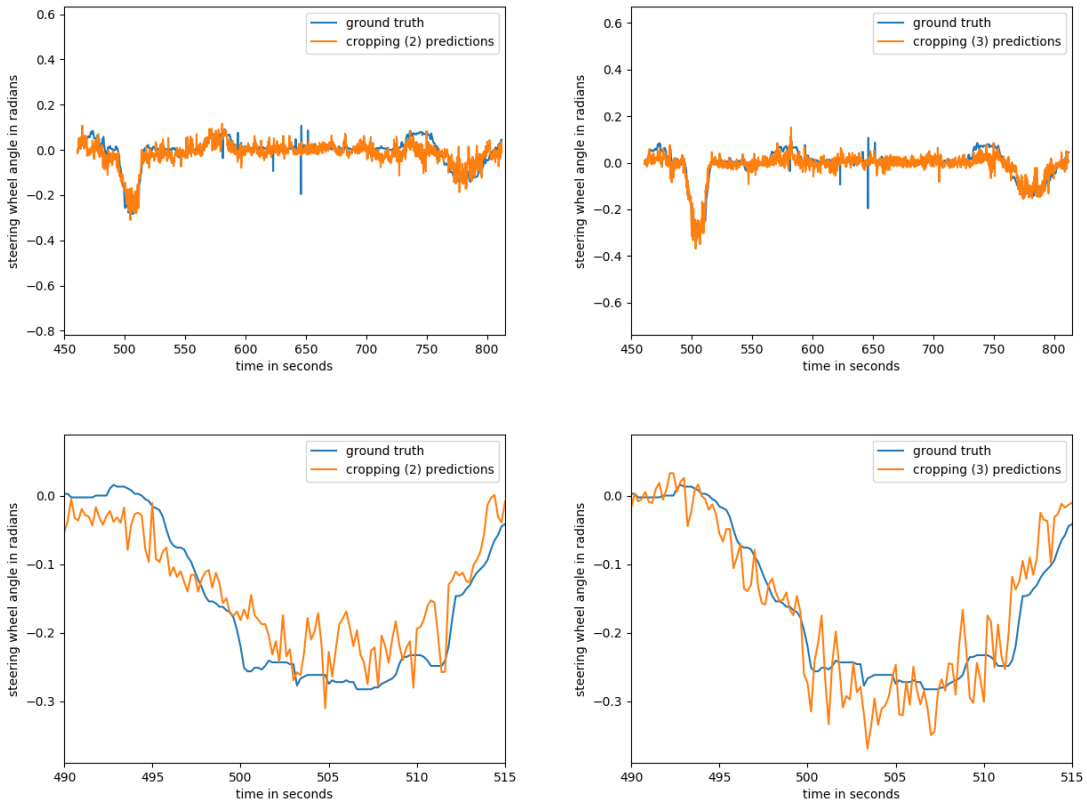


Figure 5.3: Plot of the predicted steering wheel angles for the croppings (2) (left) and (3) (right) using grayscale images for a stretch of the Berlin-Steglitz to BER trip (top) and a single curve of the same (bottom).

Results of the croppings used in conjunction with the dataset of (C) and grayscale images are detailed in Table 5.7. While obviously not reaching the same performance as on the highway dataset, the improvements achieved with the croppings are still enormous, especially considering that the usefulness of (3) should suffer from the worse-marked roads included in the complete dataset compared to the well-maintained ones found on highways.

Configuration	MSE	RMSE	MAE	MCE	Accuracy
(C) with YUV	0.01709	0.13072	0.07598	**0.03844**	0.76278
(C) with (1) and grayscale	0.01206	0.10982	0.0639	0.05161	0.8179
(C) with (2) and grayscale	0.00999	0.09997	0.06128	0.0567	0.82928
(C) with (3) and grayscale	**0.00977**	**0.09885**	**0.05555**	0.05489	**0.85817**

Table 5.7: Evaluation of the croppings on the third dataset (C).

5.7 Visualization

To get better understanding of what exactly the models managed to learn, the VisualBackProp algorithm (see section 3.8) can be used to create masks of intensity values for each pixel indicating their importance to the specific prediction. These masks were then

layered on top of the original image in the green color channel to generate visualizations of the scene features that were primarily used in the decision making process. Figure 5.5 shows the visualization for two images of the same scene, one from the grayscale model trained on highway data using cropping (2) and one with cropping (3). It is clear that both learned to recognize lane markings, especially so for cropping (3), which is not surprising, since the images hold little else. The model with cropping (2) seems to have additionally learned to identify vehicles, while ignoring irrelevant details like the buildings on the right side or the wall on the left. These results are promising, since they identify exactly those features, that a driving system would be expected to.

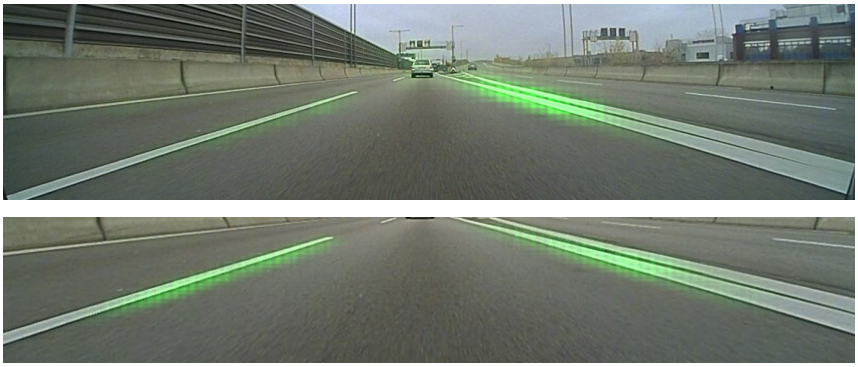


Figure 5.4: Visualization masks layered on top of the original image in the green color channel for grayscale models trained on highway data using cropping (2) (top) and (3) (bottom).

Similar visualizations for models trained on dataset (C), which includes urban driving, reveal the same. While the performance did not reach the same levels as the models exclusivley trained on highway data, they seem to be able to recognize the lane markings pretty well, while ignoring almost everything else.

Figure 5.5: Visualization masks layered on top of the original image in the green color channel for grayscale models trained on dataset (C) using cropping (2) (top) and (3) (bottom).

6 Conclusion

In this work, a general overview of end-to-end learning and the behavior reflex approach for autonomous driving has been given. The insights gained in this, have then been used to train a model to steer a car using driving data collected by the AutoNOMOS Labs project.

The problem of creating an autonomous driving model is very complex. Many papers have been written in the last several years using vastly different approaches, and a general solution is as of yet not in sight. A lot of different decisions, limitations, and pitfalls have to be considered before even the attempt can be made. Some of these have been shown and compared in this thesis.

Four different datasets have been created from the collected data, by filtering according to different criteria and considering which characteristics are desirable or detrimental when training a driving system. Different pre-processing steps including augmentation, colorspace conversion, rectification, cropping, and pruning have been examined and attempted.

A model has been trained to predict steering wheel angles from images recorded with a front view camera using an existing, relatively simple convolutional neural network that has been extended with different deep learning techniques like batch normalization, dropout, and weight decay. The model was tested with different configurations of datasets and preprocessing steps and evaluated using different metrics and visualization techniques.

In the end, the best performance has been achieved by training on exclusively highway driving, by using grayscale images cropped to exclude their top and bottom regions, by pruning the dataset to remove its bias towards driving straight and by augmenting the dataset with mirrored versions of each image. The model was shown to be able to follow the road and to shadow the ground truth relatively well when driving through curves. It was also shown, using a visualization technique called VisualBackProp, to have learned to recognize lane markings and to use them primarily to make the steering predictions.

6.1 Future Work

One of the biggest reductions compared to other works in the field, is the disregard of temporal information of the driving system. Past driving actions and sensory data are not considered, which limits the information available to make driving decisions severely. Two possible approaches to this would be to use a stack of consecutive images as input instead of a single one or to use a recurrent network module in the network architecture, like an LSTM. This would additionally allow the prediction of speed control on top of steering.

There are other promising techniques that can be used to enhance the network like pretraining some parts of the network on other, bigger datasets like ImageNet or using scene segmentation as side or auxiliary task to influence the decision making of the system. These have been shown to be very successful when training a driving system and are able to mitigate the problems of having only a small dataset.

One of the key limitations imposed in this work, that is at the same time crucial to achieve any results at all, is the removal of lane changes and intersections from the dataset. However these are obviously part of the possible scenarios encountered during real-world driving, and finding solutions that allow their inclusion would be a necessary step in creating a driving system that can do more than just follow the road. One possibility for this would be to feed route planner data to the network, to allow it to anticipate necessary lane changes or turns.

The current network predictions are bound to the geometry of the vehicle that was used for the collection of the dataset since steering wheel angles translate to different sized turns for different cars. Separating these two by transforming the steering wheel angles to geometry independent turning angles, could be a possible next step to allow the usage of other datasets collected with different vehicles. The current dataset consists of only about 13 hours of driving and is therefore relatively small. The model's performance could be improved immensely by collecting more data or using other, existing datasets. Another idea along the same lines, is to use further augmentation techniques. A very promising one is the utilization of images recorded by additional cameras to simulate different positions in the lane. Transforming the corresponding steering wheel angles to lead the car back towards the center of the lane, can teach driving system to recover from mistakes. While data from these additional cameras is only available for about half of the dataset, this might be enough for some experimentation.

The evaluation of the trained models is very limited thus far. A more conclusive examination could be undertaken with simulations or similar approaches. While the general performance on the validation set has been measured and observed for a small stretch of highway driving, a more thorough study of which parts of the dataset were problematic for the model, might give further clues on how to improve it. Additionally, the difference between night, dusk or day trips and how they would play a role for the driving system, has not been investigated in this work.

References

[1] C. Badue, R. Guidolini, R. V. Carneiro, P. Azevedo, V. B. Cardoso, A. Forechi, L. F. R. Jesus, R. F. Berriel, T. M. Paixão, F. Mutz, T. Oliveira-Santos, and A. F. De Souza (2019). Self-driving cars: A survey. *arXiv e-prints* arXiv:1901.04407.

[2] M. Bojarski, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, U. Muller, and K. Zieba (2017). Visualbackprop: Efficient visualization of cnns. *arXiv e-prints* arXiv:1611.05418.

[3] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba (2016). End to end learning for self-driving cars. *arXiv e-prints* arXiv:1604.07316.

[4] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, and U. Muller (2017). Explaining how a deep neural network trained with end-to-end learning steers a car. *arXiv e-prints* arXiv:1704.07911.

[5] C. Chen, A. Seff, A. Kornhauser, and J. Xiao (2015). Deepdriving: Learning affordance for direct perception in autonomous driving. In *IEEE International Conference on Computer Vision (ICCV)*.

[6] Y. Chen, P. Palanisamy, P. Mudalige, K. Muelling, and J. M. Dolan (2019). Learning on-road visual control for self-driving vehicles with auxiliary tasks. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*.

[7] Y. Chen, J. Wang, J. Li, C. Lu, Z. Luo, H. Xue, and C. Wang (2018). Lidar-video driving dataset: Learning driving policies effectively. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[8] L. Chi and Y. Mu (2017). Deep steering: Learning end-to-end driving model from spatial and temporal visual cues. *arXiv e-prints* arXiv:1708.03798.

[9] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy (2018). End-to-end driving via conditional imitation learning. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[10] X. Glorot, A. Bordes, and Y. Bengio (2011). Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323. PMLR.

[11] X. Glorot and Y. Bengio (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256. PMLR.

[12] I. Goodfellow, Y. Bengio, and A. Courville (2016). Deep Learning. MIT Press. `http://www.deeplearningbook.org`.

[13] S. Hecker, D. Dai, and L. Van Gool (2018). End-to-end learning of driving models with surround-view cameras and route planners. In *Computer Vision - ECCV 2018*, pages 449–468. Springer.

[14] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue, F. Mujica, A. Coates, and A. Y. Ng (2015). An empirical evaluation of deep learning on highway driving. *arXiv e-prints* arXiv:1504.01716.

[15] S. Ioffe and C. Szegedy (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv e-prints* arXiv:1502.03167.

[16] J. Janai, F. Güney, A. Behl, and A. Geiger (2017). Computer vision for autonomous vehicles: Problems, datasets and state-of-the-art. *arXiv e-prints* arXiv:1704.05519.

[17] J. Kim and J. Canny (2017). Interpretable learning for self-driving cars by visualizing causal attention. In *IEEE International Conference on Computer Vision (ICCV)*.

[18] D. P. Kingma and J. L. Ba (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*.

[19] D. Kriesel (2007). A Brief Introduction to Neural Networks. `http://www.dkriesel.com`.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.

[21] Y. LeCun, Y. Bengio, and G. Hinton (2015). Deep learning. In *Nature*. vol. 521, pages 436–444.

[22] Y. LeCun, U. Muller, J. Ben, E. Cosatto, and B. Flepp (2005). Off-road obstacle avoidance through end-to-end learning. In *Advances in Neural Information Processing Systems 18*, pages 739–746. MIT Press.

[23] F.-F. Li, J. Johnson, and S. Yeung. (2019). Stanford university cs231n: Convolutional neural networks for visual recognition. `http://cs231n.github.io/`.

[24] A. I. Maqueda, A. Loquercio, G. Gallego, N. García, and D. Scaramuzza (2018). Event-based vision meets deep learning on steering prediction for self-driving cars. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[25] A. Mehta, A. Subramanian, and A. Subramanian (2018). Learning end-to-end autonomous driving using guided auxiliary supervision. *arXiv e-prints* arXiv:1808.10393.

[26] M. A. Nielsen (2015). Neural Networks and Deep Learning. Determination Press. `http://neuralnetworksanddeeplearning.com/`.

[27] E. Ohn-Bar and M. M. Trivedi (2016). Looking at humans in the age of self-driving and highly automated vehicles. In *IEEE Transactions on Intelligent Vehicles*.

[28] D. A. Pomerleau (1989). Alvinn: An autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems 1*, pages 305–313. Morgan-Kaufmann.

[29] R. Rojas (1996). Neural Networks: A Systematic Introduction. Springer.

[30] S. Ruder (2016). An overview of gradient descent optimization algorithms. *arXiv e-prints* arXiv:1609.04747.

[31] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, vol. 15, pages 1929–1958.

[32] Wikipedia contributors. Fisheye lens. `https://en.wikipedia.org/w/index.php?title=Fisheye_lens&oldid=895836312`, last access on 07/05/2019. 2019.

[33] H. Xu, Y. Gao, F. Yu, and T. Darrell (2017). End-to-end learning of driving models from large-scale video datasets. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[34] L. Yang, X. Liang, T. Wang, and E. Xing (2018). Real-to-virtual domain unification for end-to-end autonomous driving. In *Computer Vision - ECCV 2018*, pages 553–570. Springer.

[35] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus (2010). Deconvolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[36] H. Zhu, K.-V. Yuen, L. Mihaylova, and H. Leung (2017). Overview of environment perception for intelligent vehicles. In *IEEE Transactions on Intelligent Transportation Systems*.