Horst Fuchs, BSc

Visualizing and Understanding Deep Driving Models

Master's Thesis

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Information and Computer Engineering

submitted to

Graz University of Technology

Supervisors

Dipl.-Ing. Dr.techn. Christoph Feichtenhofer, BSc
Ao.Univ.-Prof. Dipl.-Ing. Dr.techn. Axel Pinz

Institute of Electrical Measurement and Measurement Signal Processing

Graz, December 2017

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis.

Date	Signature

Acknowledgements

First I would like to thank my two supervisors Christoph Feichtenhofer and Axel Pinz for all their effort to support my research for this master's thesis. Beside answering and discussing with me various research questions, they also guided me in topics like good scientific work and scientific writing. In particular, I thank Christoph for coauthoring my first scientific paper, which evolved from the research for this thesis, and the "endless" hours we spend in perfecting this paper.

I would also like to thank the Institute of Electrical Measurement and Measurement Signal Processing for providing such a welcoming working atmosphere.

Furthermore, I want to give thanks to all my fellow students who worked and studied with me in multiple projects and for various exams through my entire university career. All the fruitful discussions and hours we spend talking about topics related to our studies helped to bring me (and I think also them) to this final step of graduating with a master's degree.

Of course I would also like to thank my other friends for supporting and sometimes distracting me a little bit in moments when I was not able anymore to see the wood for the trees. Sometimes a little break is all one needs to continue researching with new elan.

Finally, I thank my family who has supported me throughout my career and never complained when I overwhelmed them with technical details in conversations about my studies. Thanks to them I always could concentrate on reaching my goals and they have never let me down.

Abstract

This work is inspired by the striving research in the field of autonomous driving. Many major car producers currently work on solutions for robust and safe systems to enable cars navigating automatically through daily traffic situations. This development starts with driving assistance systems (*e.g.* parking assistants, brake assistant) and should lead to full automation of the car eventually. Self-driving cars have to fulfill many requirements – from safety and robustness to security and comfort – which made the deployment of such vehicles not feasible for a long time. With the ongoing success of deep learning and improvements in computer vision, modern cars are able to detect and track other traffic participants, to predict possible driving trajectories and to navigate independently. However, these systems are not yet failproof and they cannot solve complex situations. In such situations it is still necessary that a driver takes over in time.

In this thesis I implement artificial neural networks for lane following. Instead of predicting only one steering command at each time step, I present models which predict steering angles also for the future, *i.e.* a complete trajectory is predicted for each time step. Using open-source datasets I analyze how reliable these future predictions are and I contribute a method how future predictions can be superimposed to improve the steering of the car.

Autonomous cars can cause severe damages and casualties if they fail. In order that customers can build trust in self-driving vehicles, car manufactures have to make sure that they can explain how driving prediction are made and how the car reacts in unusual situations or different weather conditions. However, neural networks are not easy to debug and visualize due to their high number of parameters. Therefore, the second part of my thesis analyses current techniques how the learned knowledge of neural network can be visualized.

First, I present methods, where those regions of the input sequences are highlighted, which are salient for the steering prediction of the model by visualizing the activations in the convolutional layers of the model. Second, I show how preferred input sequences for a driving model can be generated without using any input data. For that I contribute a Spatiotemporal Generative Adversarial Network (ST-GAN) for visualizing and understanding deep driving networks. The ST-GAN generates video sequences in the color

domain using a fully convolutional network in spacetime. However, this network though restricts the temporal length of the generated video to the size of the spatiotemporal filter. I propose an approach based on an autoencoder with a future prediction module to sample long-term videos of arbitrary length. The trained ST-GAN is applied in the activation maximization framework to generate sequences which activate the selected unit maximally. The generated videos are highly intuitive and reveal strengths as well as weaknesses of the underlying systems. My experiments show that there are dead hidden units in end-to-end trained driving models, and interestingly that driving models with recurrent models do not learn temporal coherences in driving motion. Further experiments show that the ST-GAN can be conditioned to produce certain input data, *e.g*. driving through winter scenery although the driving model has never seen snow.

Visualizing and understanding deep driving models does not only build the trust of customers in the new technology, but also helps researchers and developers to improve current models and to reach the final goal of fully autonomous cars.

Contents

Abstract v

1. Introduction 1
- 1.1. Improving driving commands . . . 2
- 1.2. Visualization of what has been learned . . . 3

2. Background 7
- 2.1. Deep learning . . . 7
- 2.2. Artificial neural networks . . . 8
  - 2.2.1. Convolutional neural networks . . . 9
  - 2.2.2. Sequence-to-sequence networks . . . 9

3. Related Work 11
- 3.1. Prediction of driving commands . . . 11
- 3.2. Neural network visualizations . . . 13
- 3.3. Driving datasets . . . 15
  - 3.3.1. Udacity dataset . . . 16
  - 3.3.2. Comma.ai driving dataset . . . 20
  - 3.3.3. Princeton dataset . . . 21
  - 3.3.4. Berkley DeepDrive Video dataset . . . 21
- 3.4. Driving models . . . 24
  - 3.4.1. Single-frame regression models . . . 24
  - 3.4.2. Udacity challenge models . . . 26
  - 3.4.3. Berkley DeepDrive models . . . 29

4. Analyzing & Improving Driving Models 33
- 4.1. Methods . . . 34
  - 4.1.1. Architecture overview . . . 34
  - 4.1.2. CNN architecture . . . 34
  - 4.1.3. Future prediction approach . . . 35
  - 4.1.4. Seq2seq architecture . . . 37
  - 4.1.5. Superposition of future predictions to improve steering . . . 39
- 4.2. Experimental setup . . . 40
  - 4.2.1. Implementation details . . . 41

4.2.2. Data augmentation 42
4.2.3. Evaluation metric 43
4.3. Experiments 43
4.3.1. Baselines without future predictions 43
4.3.2. Reliablity of future predictions 45
4.3.3. Improving the prediction by combining with future predictions from previous time steps 50
4.3.4. Future predictions with different architectures 52
4.4. Conclusion 54

5. Visualizing & Understanding Driving Models 55
5.1. Methods 56
5.1.1. Visualization of embedded convolutional features 56
5.1.2. Class activation maps 57
5.1.3. Activation maximization without a learned prior 58
5.1.4. Activation maximization using a spatiotemporal GAN 59
5.2. Experimental setup 68
5.2.1. CNN feature embedding with t-SNE 68
5.2.2. Class activation maps for driving models 69
5.2.3. Activation maximization with Udacity driving models 70
5.2.4. Generating inputs using the spatiotemporal GAN 70
5.3. Experimental results 74
5.3.1. CNN feature embedding with t-SNE 74
5.3.2. Class activation maps for driving models 76
5.3.3. Activation maximization without a learned prior 81
5.3.4. ST-GAN: Short-term prediction experiments 83
5.3.5. ST-GAN: Long-term prediction experiments 85
5.3.6. ST-GAN: Visualizing continuous driving models 87
5.3.7. ST-GAN: Visualization of hidden units 88
5.3.8. ST-GAN: Conditioning on multiple models 92
5.3.9. ST-GAN: Generating driving videos from an image 95
5.4. Conclusion 96

6. Summary and Outlook 97

A. More hidden layer visualizations using ST-GAN 101

B. Unrolled video clips 109

Bibliography 133

List of Figures

1.1. Autonomous driving levels 1
1.2. Overview driving model 3
1.3. Overview for the spatiotemporal GAN visualization 4

2.1. Venn diagram for deep learning 8

3.1. Perspectives of Udacity dataset 16
3.2. Dataset samples Udacity 17
3.3. Steering angles over time (CH2_001) 17
3.4. Histogram of steering angles (CH2_001) 18
3.5. Steering angles over time (CH2_002) 19
3.6. Histogram of steering angles (CH2_002) 19
3.7. Steering commands over time (CH2_001) 20
3.8. Example image of TORCS dataset 21
3.9. Samples of BDD-V dataset 22
3.10. Steering command distribution of BDD-V (discrete) 22
3.11. Steering angle distribution of BDD-V (continuous) 23
3.12. Architecture of the Nvidia model 25
3.13. Driving model architecture with ResNet-50 25
3.14. Model overview team Rambo 27
3.15. CNN architecture team Chauffeur 28
3.16. Description output neurons (BDD continuous model) 29
3.17. CNN and CNN-LSTM model architecture 30
3.18. FCN-LSTM and CNN-LSTM model architecture 31

4.1. Architecture overview 34
4.2. Overview of seq2seq approach with augmentation. 36
4.3. Seq2seq model architectures Φ_{simple} and Φ_{pro} 38
4.4. Superposition principle of future predictions 39
4.5. Superposition with a neural network 40
4.6. Visualization of the predictions (single-frame models) 45
4.7. Future prediction errors (near-range) 46
4.8. Future prediction errors (middle-range) 48
4.9. Future prediction errors (long-range) 49

4.10. Steering trajectory and superimposed prediction visualization 52

5.1. Training princpile ST-GAN 59
5.2. ST-GAN architecture 62
5.3. Long-term video generation 63
5.4. Temporal convolution approach 64
5.5. Training framework seq2seq modules 66
5.6. Spatiotemporal activation maximization 67
5.7. Seq2seq model architecture for long-term generation 72
5.8. Features visualization using t-SNE. 75
5.9. Colormap for heatmaps to visualize the attention on pixels. 76
5.10. CAM single-frame input 76
5.11. CAM Chauffeur-seq2seq 77
5.12. Comparison of CAM snippets 78
5.13. CAM results Nvidia model 79
5.14. CAM results continuous BDD model 80
5.15. CAM results discrete BDD models 81
5.16. AM of first Chauffeur CNN layers 81
5.17. AM of final Chauffeur CNN layers 82
5.18. AM of Chauffeur CNN layers initialized with images 83
5.19. AM FCN-LSTM four output layers 83
5.20. AM short-term visualization 85
5.21. AM long-term generation 86
5.22. Comparison of long-term generation video snippets 86
5.23. AM continuous models 87
5.24. AM hidden LSTM features 88
5.25. AM hidden fc7 features 89
5.26. AM hidden fc6 features 90
5.27. AM hidden conv5 features 91
5.28. AM hidden conv4 features 91
5.29. AM hidden conv3 features 92
5.30. AM hidden conv3 features 92
5.31. Overview AM with scene model 93
5.32. Image samples from MIT places dataset 93
5.33. AM with scene model 94
5.34. AM initialized with image 95

A.1. AM video generation LSTM layer 102
A.2. AM video generation fc7 layer (dark) 103
A.3. AM video generation fc7 layer (daylight) 104
A.4. AM video generation fc6 layer (dark) 105
A.5. AM video generation fc6 layer (daylight) 106
A.6. AM video generation conv5 layer 107

A.7. AM video generation conv4 layer 107
A.8. AM video generation conv3 layer 108
A.9. AM video generation conv2 layer 108

B.1. Unrolled video: four directions 110
B.2. Unrolled video for S-GAN: CNN 111
B.3. Unrolled video for S-GAN: CNN-LSTM 112
B.4. Unrolled video for S-GAN: FCN-LSTM 113
B.5. Unrolled video for ST-GAN: CNN 114
B.6. Unrolled video for ST-GAN: CNN-LSTM 115
B.7. Unrolled video for ST-GAN: FCN-LSTM 116
B.8. Unrolled video: short-term approach 117
B.9. Unrolled video: long-term temporal convolution approach 118
B.10. Unrolled video: long-term seq2seq approach 119
B.11. Unrolled video: past conditioned long-term seq2seq approach 120
B.12. Unrolled video: continous BDD model 121
B.13. Unrolled video: continous ResNet-50 model 122
B.14. Unrolled video: continous Nvidia model 123
B.15. Unrolled video: hidden fc6 features (daylight) 124
B.16. Unrolled video: hidden fc6 features (dark) 125
B.17. Unrolled video: scene model (1) 126
B.18. Unrolled video: scene model (2) 127
B.19. Unrolled video: scene model (3) 128
B.20. Unrolled video: scene model (4) 129
B.21. Unrolled video: scene model (5) 130
B.22. Unrolled video: image initialization (highway) 131
B.23. Unrolled video: image initialization (blue-gray drawing) 132

1. Introduction

The development of self-driving cars is a recent research topic all over the world. The progress in autonomous driving is categorized in five levels [91] as shown in Fig. 1.1. Currently manufactures of cars already ship cars with driver assistance systems of level 2 and 3, where the car takes over the steering and accelerating/breaking. However, the driver still has to take over the control when the system requests it. The research goal is to develop a robust and reliable system which fulfills all requirements for level 5: Full Automation. In level 5 there is no driver required anymore – the system takes over the complete driving task.

SAE level	Name	Narrative Definition	Execution of Steering and Acceleration/ Deceleration	*Monitoring* of Driving Environment	Fallback Performance of *Dynamic Driving Task*	System Capability *(Driving Modes)*
***Human driver* monitors the driving environment**						
0	**No Automation**	the full-time performance by the *human driver* of all aspects of the *dynamic driving task*, even when enhanced by warning or intervention systems	Human driver	Human driver	Human driver	n/a
1	**Driver Assistance**	the *driving mode*-specific execution by a driver assistance system of either steering or acceleration/deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	Human driver and system	Human driver	Human driver	Some driving modes
2	**Partial Automation**	the *driving mode*-specific execution by one or more driver assistance systems of both steering and acceleration/ deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	**System**	Human driver	Human driver	Some driving modes
***Automated driving system* ("system") monitors the driving environment**						
3	**Conditional Automation**	the *driving mode*-specific performance by an *automated driving system* of all aspects of the dynamic driving task with the expectation that the *human driver* will respond appropriately to a *request to intervene*	System	**System**	Human driver	Some driving modes
4	**High Automation**	the *driving mode*-specific performance by an automated driving system of all aspects of the *dynamic driving task*, even if a *human driver* does not respond appropriately to a *request to intervene*	System	System	**System**	Some driving modes
5	**Full Automation**	the full-time performance by an *automated driving system* of all aspects of the *dynamic driving task* under all roadway and environmental conditions that can be managed by a *human driver*	System	System	System	**All driving modes**

Copyright © 2014 SAE International. The summary table may be freely copied and distributed provided SAE International and J3016 are acknowledged as the source and must be reproduced AS-IS.

Figure 1.1.: Levels of autonomous driving from [91].

This thesis deals with neural networks which are trained on predicting the possible steering command or ego-motion. There are two major approaches for teaching a driving model how to control the car. The first approach implements a two-stage model, where a) hand-crafted features are extracted, and b) the steering commands are predicted from

these features [10, 64]. These hand-crafted features help to abstract the real world and to simplify the prediction of driving commands. But is it really that easy? The extracted features might not be detailed enough for a deliberate prediction or edge cases might not be covered. The second approach is to train the feature extraction and the steering command prediction in an end-to-end fashion without manually defining features. That means the model can learn what features are important for a prediction.

In 2016, a a research team of NVIDIA has shown that end-to-end learning of driving commands is possible [6]. They use a neural network trained to map from input image frames of a camera behind the windshield to the steering command, the steering angle. Instead of designing hand-crafted features, they give the model the freedom to learn its own representation of the environment. However, this approach has the drawback that it needs lots of training data. This can be achieved by augmenting the available data.

Problem statement: This thesis addresses two major issues in autonomous driving: a) improving existing driving models (Chapter 4), and b) understanding driving models through visualization (Chapter 5). For improving driving models, I work on better predictions for models which are trained on lane following where the steering angle is predicted conditioned on monocular camera frames. For understanding driving models I evaluate current visualization techniques for highlighting salient image features and for generating the preferred input of the driving model. A model for generating an image which maximally activates a particular unit of a visualized model has been shown in [74, 75]. However, my research focuses on the generation of driving videos with arbitrary length and high resolution.

1.1. Improving driving commands

Inspired by the success of Nvidia a competition to train a model for lane-following was launched by Udacity for the online course "Self-driving car engineer" [9]. In this master's thesis I present in Sect. 3.4.2 the winning approaches of the Udacity Challenge (Sect. 3.4.2) and I propose an extension with a sequence-to-sequence module (Sect. 4.1.3) to predict also future steering commands, *i.e.* the self-driving car is able to drive foresightfully. I show in Chapter 4 that a driving model that looks into the past and the future improves the performance. An overview of my approach is shown in Fig. 1.2, where a convolutional neural network extracts the features from a video sequence. Conditioned on the sequence of features the sequence-to-sequence model predicts the steering commands.

Please note that the proposed system is only one more step towards full automation. Lane following is only one part of the driving task which consists of many more pieces of the puzzle. Another important task is for example the path finding to navigate through

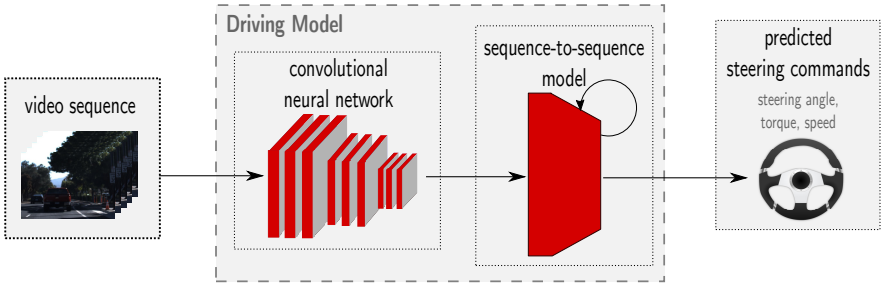


Figure 1.2.: Future prediction approach with sequence-to-sequence model.

the environment and to reach a destination. In this thesis I focus on improvements in lane following by imitation learning on data from California, USA, as described in Sect. 3.3. The neural network observes a human driver to learn controlling the vehicle. The requirements for a self-driving car are robust steering predictions that are tolerant to different driving conditions (*e.g.* changing weather conditions or different times of day). Therefore, in the second part of my thesis I visualize the learned model parameters of trained neural networks to understand why a certain steering command is predicted.

1.2. Visualization of what has been learned

Nowadays neural networks are widely used as state of the art models to solve perception tasks. Beside approaches for semantic segmentation [33, 107] and tracking objects [103, 107] in driving scenes, also learning of driving commands from video frames has shown significant improvements. Nevertheless, before deploying the models for large-scale production of self-driving cars, customers and industry partners first want to know what a neural network has learned. Treating the neural network as a black box is no option; understanding of the learned parameters is essential, for example to explain why an accident has happened or how the self-driving car would react in a certain environment. Furthermore, the visualization and understanding of the neural network also helps to improve and extend its functionality.

The second research part of my thesis (Chapter 5) shows approaches for visualizing neural networks using two different ideas: highlighting areas in the input frames which are salient for the prediction [96], and generating preferred input images for the driving model.

In Sect. 5.1.2 I use class activation maps [95, 132] to visualize which features in the input frames attract the attention of the driving models. With this method one can debug a neural network by looking if the model focuses on meaningful features, *e.g.* road markings, road signs, or brake lights of other cars. However, this analysis is always based on the quality of the available input data. Therefore, I implemented another technique based on activation maximization (AM) [77, 29, 30], where the preferred input of a model is generated by maximizing the activation of a certain prediction.

However, in Sect. 5.1.3 we see that the generated images are tricky to interpret for a human, because the generation of natural images needs a strong prior [76, 77].

Improving natural image generation has been already tackled for other domains, like dynamic texture generation [106, 24]. Recently, Generative Adversarial Networks (GANs) for image generation [38, 87, 22] have shown that they can learn to produce images in a unsupervised manner. In this thesis I present the spatiotemporal GAN (ST-GAN) for video generation to visualize deep driving models. The research about the ST-GAN in the context of autonomous driving has been submitted for publication at the conference of computer vision and pattern recognition (CVPR2018) in collaboration with Christoph Feichtenhofer under the title "ST-GANs for Visualizing and Understanding Deep Driving Models".

The development of ST-GAN is based on the research of Dosovitskiy *et al*. [25], who used a GAN for learning image generation from unlabeled image data. We use this way of supervision to train the ST-GAN additionally on the temporal coherence in videos. After training the ST-GAN we use AM [29, 96, 74] to generate preferred input videos for deep driving models. Our implementation is a modification of the Plug&Play Generative Networks (PPGN) framework of Nguyen *et al*. [75], which we extend for video generation by using our ST-GAN.

Video generation is harder than image generation due to the additional challenges of motion and scene dynamics. Therefore many approaches only generate short videos of low resolution [92, 115]. We propose a recurrent future prediction module to generate long sequences of arbitrary duration and high resolution (Sect. 5.1.4). Additionally, we also experiment with changing the scenery of the driving scene by conditioning on multiple models or initializing the input data of the AM framework non-randomly.

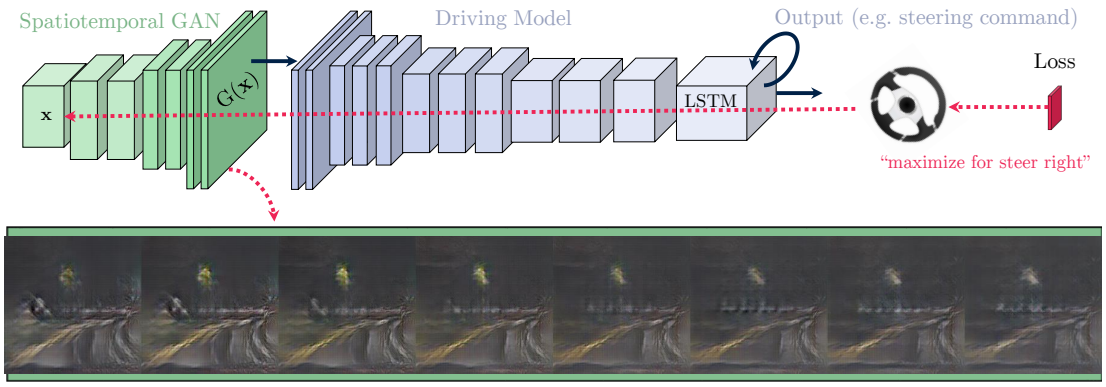


Figure 1.3.: Our spatiotemporal GAN generates preferred input video sequences for a condition network. We use driving models as condition network, which are trained to predict the steering angle from monocular videos. In the shown video sequence, we notice the yellow road markings and the moving lights of an oncoming car on the other lane.

Figure 1.3 shows the overview for the spatiotemporal visualization with ST-GAN, where we inspect a driving model from [124] that is trained on human driving imitation. The model pushes a sequence of image frames (*i.e.* a video sequence) through a convolutional network and an LSTM that maps the visual input to a steering command prediction. The image frames below show 8 frames generated with our ST-GAN that maximally activate the steering command 'right' of the driving model. The video generation is initialized with Gaussian noise and no input images are used to generate our visualizations.

In comparison to the simple AM without a prior, the generated videos from our ST-GAN are easy to interpret by a human. In the results we show visualizations of different driving models (Sect. 5.3.4 – 5.3.6), several hidden layers (Sect. 5.3.7), and experiments where we change the driving scenery (Sect. 5.3.8 – 5.3.9).

2. Background

This chapter provides a short overview on related background, with a brief explanation of artificial neural networks. In the latter I present the convolutional neural network and the sequence-to-sequence model, which I use as models in this thesis. This chapter is based on the introduction of the book "Deep Learning" of Goodfellow *et al*. [37].

2.1. Deep learning

Creating artificial intelligence (AI) algorithms has been already a research field in the past century. Many approaches tried to formulate tasks and problems as a set of rules and conditions. In **knowledge based algorithms** the machine can infer based on the input data and the hard-coded program logic. For many computer vision tasks, including the driving task of autonomous vehicles, it is not feasible to define the environment with hand-crafted rules. Therefore, an automatic extraction of patterns from raw data is necessary. An algorithm with the capability to learn these patterns is called **machine learning algorithm**. Examples for machine learning algorithms are logistic regression and naive Bayes [73].

Machine learning algorithms learn how to make predictions by modeling a data representation consisting of features. These features can be hand-crafted (*e.g*., using the current barometric pressure as a feature for weather forecast), or the algorithm discovers features during the training process. This learning of feature extraction is called **representation learning** and it is useful for tasks, where it is not easy to describe features. One example where this is a problem: How would you describe a car in an image? Although most humans know how a car looks like, it is difficult to describe it in all details for a computer program. In representation learning the algorithm learns a number of features, which enable the model to output correct predictions.

For the above example of the image with the car, it can still be difficult to detect the car due to different lighting conditions, different perspectives, different car types, or other variations. A technique called **deep learning**, which learns data representations

in multiple hidden layers, beginning with simple representations and ending with high-level features, is effective for solving these problems. For images that means that low levels mostly learn to detect colors and edges, whereas high level filter describe contours and object parts. The basic example for a deep learning algorithm is the multilayer perceptron (MLP), a simple feed-forward neural network with at least three nodes. Goodfellow *et al.* [37] describe deep learning as a subclass of AI as shown in the Venn diagram in Fig. 2.1. There are many other techniques beside deep learning for creating AI algorithms. However, with increasing computing power and the availability of big datasets deep learning is currently a hot topic in the research community.

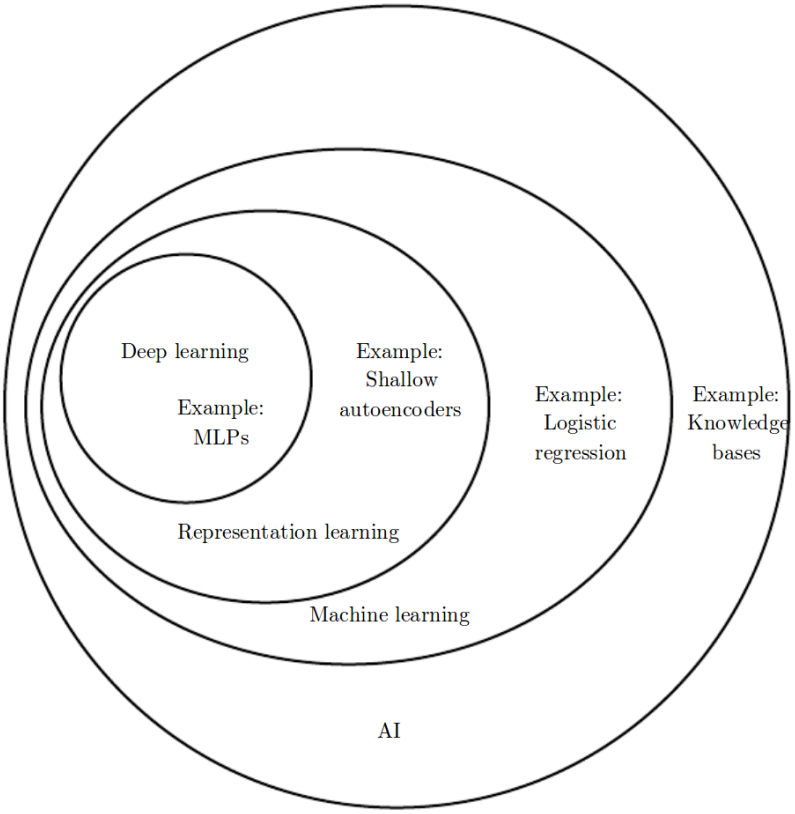


Figure 2.1.: Venn diagram from [37] showing deep learning as a small subset of AI approaches. It is a specialization of representation learning, which is a type of machine learning. Machine learning itself is a popular AI technology, but not the only one. The diagram lists examples for each group.

2.2. Artificial neural networks

Artificial neural networks (ANN) are inspired by research in the neuroscience. However, ANNs are much simpler than a biological brain and also the functional principle differs. This section describes briefly two ANN architectures which are used in most implemented models in this thesis.

2.2.1. Convolutional neural networks

Convolutional neural networks (CNN) have been introduced by LeCun *et al.* [59]. This type of ANN is particularly used to process high-dimension data like images. Basically, a CNN is a simple neural network that uses convolution instead of general matrix multiplication in one or several layers [37]. The input of the layer is convolved with a kernel, whose parameters are learned in the training procedure.
In case of input images the kernel K is two-dimensional and the convolution is defined as

$$X_{out}(i,j) = (X_{in} * K)(i,j) = \sum_m \sum_n X(i-m, j-n)K(m,n) \tag{2.1}$$

where X_{in} and X_{out} denote the two-dimensional layer input and output.

With the increasing popularity of deep learning, also the CNNs became deeper and more complex. Deep architectures like VGG [97], inception networks [104] or networks with residual connections [42] have shown impressive improvements on many challenges (e.g., the ImageNet Large Scale Visual Recognition Competition [90]). Training these deep CNNs is very expensive in time and computing resources. Therefore, transfer learning with pretrained model parameters is a popular method. For natural images the trained network parameters of popular models (*e.g.* AlexNet [56], ResNet-50 [42]) are available online, pretrained on the ImageNet dataset [90]. Finetuning on the target dataset can improve the results and for many cases it is less expensive in computing resources than training the model from scratch.

2.2.2. Sequence-to-sequence networks

Sequence-to-sequence [102] or encoder-decoder networks [16] have been developed independently by two research groups at the same time and consist of two recurrent neural networks (RNN) [44]. The input sequence $\mathbf{x}^{(1:n_x)}$ is processed by the encoder which computes a context vector C (final hidden layer of the encoder). This context vector is used as input for the decoder which models the output sequence $\mathbf{y}^{(1:n_y)}$.

This model can be trained end-to-end, and, compared to other RNNs, this architecture is able to use pairs of input and output sequences with different lengths, *i.e.* $n_x \neq n_y$. Additionally, there are no constraints that the encoder and decoder must have the same number of units in the hidden layers. The number of units can vary for encoder and decoder, and it is also possible to use several hidden layers for encoder and/or decoder.

In this thesis the layers of the sequence-to-sequence models are implemented using Long Short-Term Memory (LSTM) cells [43]. However, any recurrent cell can be used for the encoder and decoder.

Sequence-to-sequence models are applied for example for image question answering [68], image caption generation [114, 125], and video to text translations [112].

3. Related Work

This chapter presents related work to driving command predictions (Sect. 3.1) and to neural network visualizations (Sect. 3.2).

For teaching a neural network how to control a car, datasets are needed which also contain the measured driving commands, like steering angle/speed or ego-motion of the vehicle. Recently, datasets have been published that fulfill this criterion. In this chapter I review in Sect. 3.3 four datasets (Udacity, Comma.ai, Princeton, and Berkley DeepDrive) for learning steering commands.

This chapter also presents the deep driving models (Sect. 3.4) which are used in the experiments for improving the driving models (Chapter 4) and for understanding of driving models (Chapter 5).

3.1. Prediction of driving commands

For autonomous driving the prediction of steering commands must be conditioned on multiple environmental factors. Whereas various projects deal with semantic segmentation [2, 20, 57] or detection [10, 13], it is also necessary to develop models which incorporate path planning [119], so the vehicle knows where to go at a crossing. On a miniaturized scale using model cars Codevilla *et al*. [19] send simple direction commands via a remote control to navigate the car through crossings. This car operates – beside the path planning – autonomously with a monocular camera. However, in this thesis I implement models trained on the task of lane following without any path planning.

Learning the steering commands from monocular video is of current interest in the research community of computer vision and machine learning. Chen *et al*. [10] describe two major types of learning to map the camera frames to the steering commands: First, the *mediated perception approach* uses at least two stages of data processing, *i.e*., the first stage learns hand-crafted features (*e.g*. edge detection of road markings and roadsides) and the following stages predict the steering commands conditioned on these

features [64, 10, 26]. Second, the *behavior reflex approach* trains the driving model in end-to-end fashion. That means there is no manual feature selection; the camera frames are mapped directly to the steering commands. This thesis focuses on this approach and below I describe several projects working with models trained end-to-end.

End-to-end learning with CNNs: Inspired by the Autonomous Land Vehicle in a Neural Network (ALVINN) [84], Bojarski *et al*. [6] implement a deep CNN with fully-connected layers attached, which maps the camera frames to a regressive output unit which represents the steering angle. They report that the model is able to learn driving by training it with the input of three cameras used for heavy data augmentation. Furthermore, they show visualizations that the model has learned to detect useful image features (*e.g*. roadsides and road markings) [7, 5].

A similar approach with three cameras is used in the work of [36]. A micro aerial vehicle is trained to find and follow trails in forests. Instead of regression to determine the flying direction, the model uses a classification to choose one of three directions (turn left, turn right, or go straight ahead). After the training the system is able to follow trails comparably to human control of the drone. Although the vehicle can navigate on wide trains, there are still problems in zones where obstacles narrow the path.

The lane keeping assistance model in [46] uses imitation learning with only one camera to train a CNN for staying in the center of the lane. Beside the performance for lane keeping, they also evaluated how comfortable it would be to sit in this car, *i.e*. if undesired jitter appears in the steering commands.

The end-to-en trained MultiNet model [107] learns simultaneously to detect other cars and to segment the road. The Berkley DeepDrive model [124] also uses semantic segmentation as side task to improve the prediction of the ego-motion of the car. Whereas the convolutional layers only learn the spatial information, this model also uses a recurrent layer to model the temporal coherence in driving.

Learning temporal structure: Recurrent layers on top of the convolutional layers are popularly used for visual recognition and description [23]. Recurrent layers enable the model to understand video sequences. That is an advantage, because models that only learn spatial image features via imitation learning lack the knowledge that also smooth trajectories are an important part of steering a car. Multiple neural networks with LSTM layers show that they are able to understand the temporal context of driving and that they improve the driving command predictions [15, 28, 27]

An extension of these models are sequence-to-sequence models which predict more than one output, *i.e*. they predict a trajectory of steering commands for the next time steps. Lee *et al*. [61] proposes a sample generation module that predicts a set of possible trajectories, which are then refined in a second stage. Hou *et al*. [45] introduce the

Fast Recurrent Fully Convolutional Network with an LSTM encoder-decoder module for predicting future driving commands. These models are similar to the sequence-to-sequence models I present in this thesis.

With LSTM layers a model already possesses short-term memory. However, in the driving task it might be useful to have access to memories which date back for longer time, *e.g.* remembering speed limits. One solution could be to use memory networks [120, 101]. This models are already deployed to reinforcement learning tasks, *e.g.* finding an item in a Minecraft 3D world [79]. Fernando *et al.* [32] show that it is also possible to use memory networks in the domain of autonomous driving.

Cognitive mapping and reinforcement learning: Differently to the presented networks, [12] proposes a model which is inspired by the human visual cortex. It uses a cognitive map and an attention mechanism to detect free-space in images recorded by three front-facing cameras mounted on a car. The model is also able to detect lines on the streets and plan the vehicle trajectories. Also Gupta *et al.* [41] uses cognitive mapping and planning for navigating a robot through the environment only based on the visual input. Although the model works well in simulations, in real world a perfect odometry would be necessary to achieve the same results.

Instead of predicting the next steering commands, Santana and Hotz [94] model a network for video prediction that could then be used for the prediction of turns. They implement a variational autoencoder [54] using generative adversarial networks [38]. Instead of learning end-to-end, first the autoencoder is trained and then the produced features are used to train a recurrent neural network.

So far all described models use supervised learning. Deep Reinforcement Learning (RL) is another way to teach model how to drive. Instead of supervising the training, the model gets rewards if it achieves certain goals. Beside positive reward also negative rewards can be modeled to penalize the algorithm for bad predictions. RL has already shown good performance in other domains (*e.g.* playing Atari games [70]), but for the self-driving car problem there are still open challenges. Simple RL in real world on a road is not feasible – imagine a car learning how to drive by trial and error – therefore, this algorithm has been implemented in simulators [71, 34] .

3.2. Neural network visualizations

Network interpretation: Different approaches tackle how to visualize what a neural network has learned. The filters of the convolutional layers can be visualized directly by recording the activation and sampling over the dataset to find the input data which maximizes the interesting features [128, 131, 132, 3], but it is not easily understandable

how a combination of these filters lead to a certain prediction. Saliency maps or class activation maps [5, 7, 96, 130, 99, 95] highlight pixels in the input image that contribute to the obtained prediction. An improvement is the visual attention model with causal filtering to highlight deciding regions in the input frames [52]. The drawback of all these methods is that they always need input data to analyze.

Activation maximization (AM) generates the preferred input for a unit from noise [127, 67, 81, 72]. Before analyzing and generating video sequences the generation of natural and realistic images has been studied [82, 58, 25, 39, 87, 54, 22, 117] where the goals are photo-realistic resolution [60, 51] and sampling of diverse images [29, 77, 118].

Generating images with activation maximization without a strong image prior revealed that neural networks can be "fooled" with unnatural input images which cannot be perceived as the input image for a certain output class by a human [105, 76]. This suggests, that a strong prior is necessary to generate natural input images. This can be achieved by training a generative adversarial network (GAN) with unlabeled image data [25].

Beside generative models that generate images with low resolution [78, 93, 38] or are costly in computation [82, 83], the Deep Generative Network (DGN) [74] generates high-resolution images by optimizing in a latent feature space instead of the image space. Compared to the large image space, the feature space is much smaller and the generation of natural images improves. However, this approach lacks the generation of diverse images. Unifying the knowledge of [96, 127, 77, 74] the DGN is extended with an autoencoder feedback loop which enables the algorithm to sample diverse images maximized for a multi-faceted unit [75].

A two stage model with a refinement module [129] can synthesize high-quality images by training with a GAN. The implementation of perceptual loss functions for coarse-to-fine refinement enables models to synthesize high-resolution images [35, 8, 60, 49, 11]. Other applications where GANs produce high-resolution natural images are image-to-image translation [47, 136], creation of parameterized avatars [122], artistic style transfer [89, 62], and conditioning image generation on text [88, 21] or feature units [25, 74].

Generative video modeling: Video generation dynamic scenes has been used already for texture generation [106, 24]. Nowadays, due to the availability of more data and computing resources, deep models can be deployed for video generation. In the video space deep generative models like the Video Pixel Network [50] or GANs show significant improvements in generating video sequences. GANs can predict future video frames conditioned on multiple past video frames [116, 126, 100, 80, 50, 132, 69, 135, 65] or a single input frame [123]. The generation of motion in a video can also be

learned independently from the content [109, 113]. Another idea is to decouple static background and moving foreground in the video generation [115].

A model where background and foreground are handled separately [115] does not work for autonomous driving because of the strong ego-motion in driving scenes. Nevertheless, it is possible to train two different generators, where one learns to model temporal coherences and the other is an image generator responsible to produce natural image features [92].

3.3. Driving datasets

The field of autonomous driving is of great interest in the computer vision and machine learning research community. Therefore, multiple datasets with driving data are available. However, not all datasets for autonomous driving are suitable to teach an artificial neural network how to control a car. Large datasets like Coco [63] are designed to train neural networks for example on segmentation tasks. To teach a car how to navigate based on monocular RGB images, the model needs measurements of the steering angle synchronized with the recorded video frames. Additional measurements, like the speed of the car, can also provide improvement in the training of the artificial neural network. Table 3.1 shows an overview of available datasets, which provide the necessary data for training neural networks on the driving task.

In Chapter 4 I improve driving model which are exclusively trained and evaluated on the Udacity dataset [110]. Therefore, I analyze the Udacity dataset of Challenge 2 to give insights what properties this dataset has, *i.e.* the section provides necessary details about the recorded steering angles in the training and test set, as well as an analysis of the measured speed and the measured torque. In Chapter 5 I use the BDD-V dataset to train the spatiotemporal GAN, therefore also this dataset is briefly analyzed.

Table 3.1.: Datasets with RGB footage and measured steering commands.

Dataset	**Size** in GB	**Duration** in hours	**#cameras**	**real-world**
Udacity Driving Data	79	5.9	3	✓
Comma.ai Driving Dataset	80	7.3	1	✓
Princeton DeepDriving Dataset	50	12.0	1	
Berkley DeepDrive Video Dataset	600	$\approx 10,000$	1	✓

3.3.1. Udacity dataset

This dataset was initially created for the second challenge in the *Self-Driving Car Engineer Nanodegree* of Udacity [110]. After the end of the challenge more data with a total size of 223 GB has been published. However, the dataset has been furthermore processed and compressed, which has reduced the size of the revised dataset to 79 GB.

An overview of the datasets of Challenge 2 and 3 is shown in Table 3.2. The data was collected with a Lincoln MKZ in Mountain View, California, and it is stored in data containers (*rosbag*) of the Robot Operating System (ROS) [86].

The dataset provides among other sensor data:

- RGB video frames from front-facing cameras (left, center, and right),
- log data (brake, throttle, steering angle, and speed).

The outer cameras are turned $30°$ to the left/right w.r.t. the center camera. An example of recorded frames of all three cameras is shown in Fig. 3.1. More samples of the center camera can be found in Fig. 3.2. The stored video frames have a size of 640×480 pixels. Notably, the video sequences from Challenge 2 contain already processed data for learning lane-holding, *i.e.* the car does not stop and the sequence does not contain turns into other roads.

Table 3.2.: Four different tracks of the Udacity Dataset. CH2_002 and CH2_001 were used for the Udacity Challenge 2. CH3_002 and CH3_001 are data recordings for the Udacity Challenge 3 and contain also sections where the car stops.

Name	**Size** in GB	**Duration** in min:sec	**Details**
CH2_002	4.4	28:14	Training data for Challenge 2
CH2_001	0.5	4:40	Test data for Challenge 2
CH3_002	60.1	220:19	Down/back trip on El Camino Real
CH3_001	14.4	100:11	North- and Southbound on El Camino



Figure 3.1.: The Udacity dataset [110] is recorded using three front-facing cameras. This gives three different perspectives for each recorded time step in the driving sequence.



Figure 3.2.: Samples of video frames from the Udacity dataset [110] recorded with the center camera.

Analysis of the Udacity dataset

We use the data of the Udacity Challenge 2 for our experiments to predict driving commands. This section provides an analysis of this dataset.

The driving data for the Udacity Challenge 2 consists of two datasets: `CH2_001` (with one track, duration 4 minutes 40 seconds) and `CH2_002` (consists of four tracks, total duration 28 minutes). The dataset `CH2_001` has been used for the challenge as test set and we also use it for this purpose. Fig. 3.3 shows the steering angles over time of this test set. A steering angle α of zero means "straight ahead". A positive value

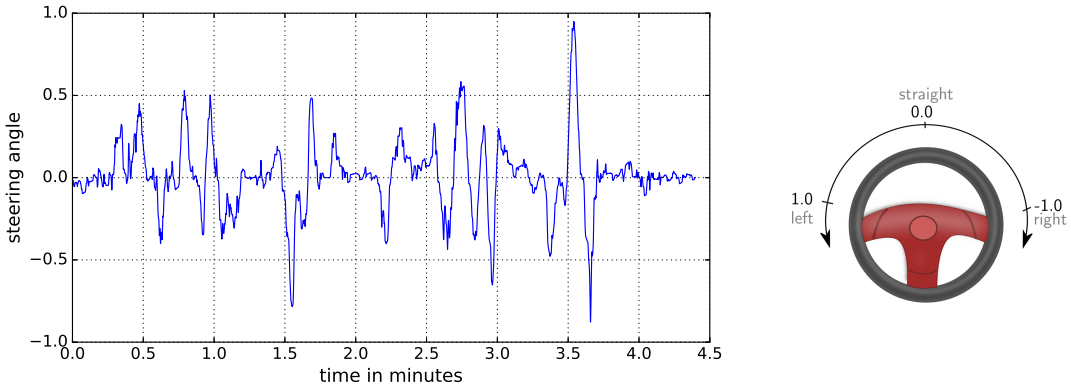


Figure 3.3.: Measured steering angles over time of the Udacity dataset `CH2_001`. Positive steering angles mean a left turn; negative ones denote a right turn. The vehicle drives straight ahead when the angle is zero. This dataset is used as test set for the models in Chapter 4.

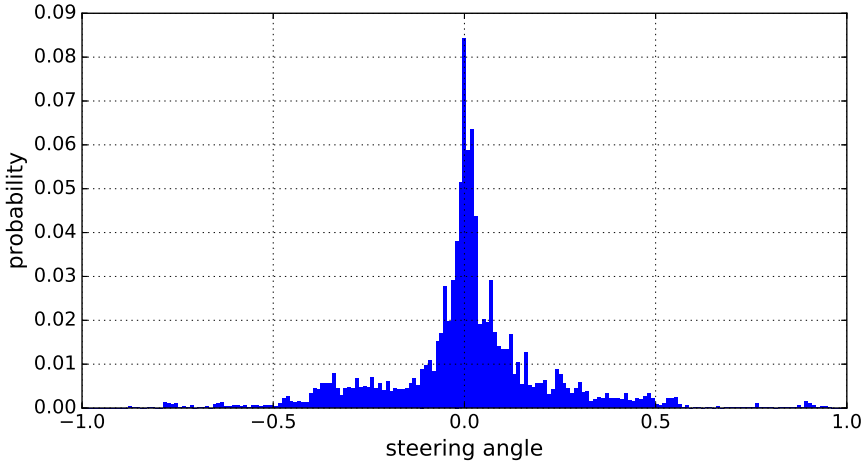


Figure 3.4.: Histogram of the steering angles of the Udacity dataset `CH2_001`. The calculation of the histogram uses 200 bins to discretize the continuous steering angle. Each bin displays the probability of the steering angle in the dataset. The histogram shows that small steering angles dominate, but also sharp turns exist.

for the steering angle denotes a left turn, whereas a negative one denotes a right turn. The distribution of the steering angles in Fig. 3.4 shows that small steering angles ($-0.15 < \alpha < 0.15$) dominate in the dataset. Furthermore, the distribution and also the plot over time shows some sharp turns in the test set.

The dataset `CH2_002` is used as training and validation set. The histogram in Fig. 3.6 shows a similar distribution to the one of the test set in Fig. 3.4. The majority of steering angles is close to "driving straight", but there is a significant amount of sharp turns. This can also be seen in the plot over time (Fig. 3.5), which shows sharp turns appearing on the route.

Both datasets do not contain sequences where the car stops and the car always stays on the same lane. Thus, the neural network only has to learn to keep the car in the lane. The datasets contain streets with direct sunlight, light changes and shadows. Some routes show two-lane roads, other parts have been recorded on multi-lane highways.

The datasets are stored in the data format *rosbag*. The tool *Udacity Driving Reader*[1] is used to extract the camera frames and the sensor measurements. The sensor measurements are synchronized with the camera frames and stored in a csv-file.

The sampling rate of the Udacity datasets is $f_s = 20Hz$.

[1] Available at: `https://github.com/rwightman/udacity-driving-reader`

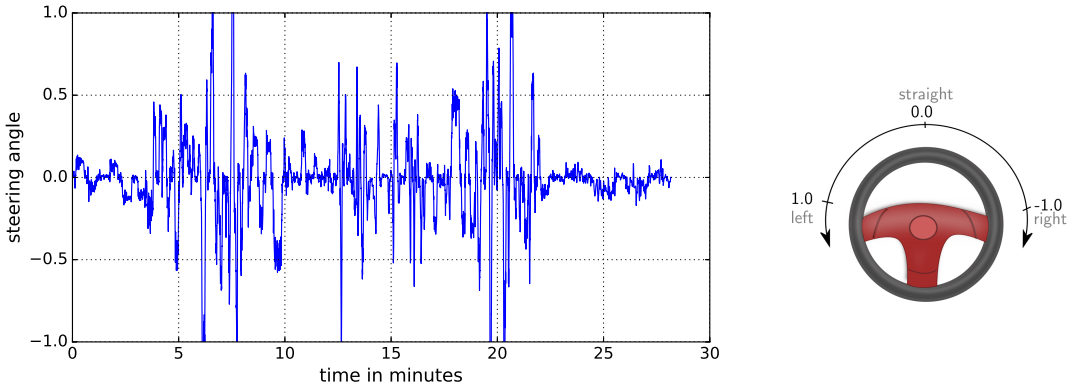


Figure 3.5.: Measured steering angles over time of the Udacity dataset `CH2_002`. Positive steering angles mean a left turn; negative ones denote a right turn. The vehicle drives straight ahead when the angle is zero. This dataset is used as training set for the models in Chapter 4.

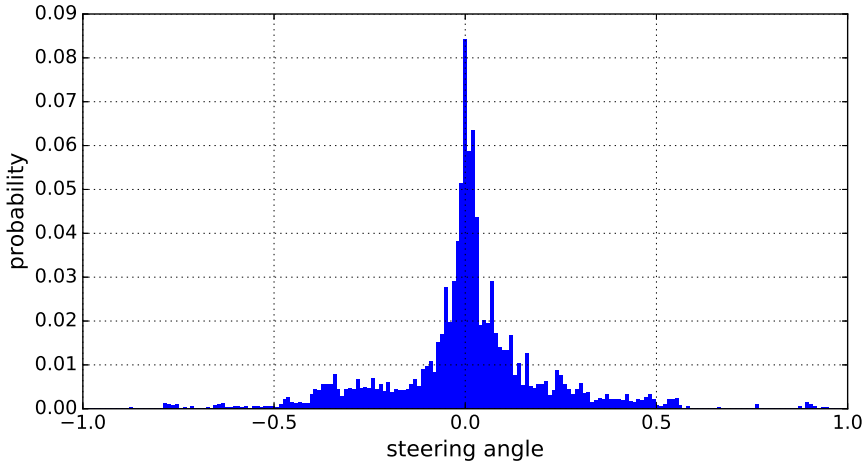


Figure 3.6.: Histogram of the steering angles of the Udacity dataset `CH2_002`. The calculation of the histogram uses 200 bins to discretize the continuous steering angle. Each bin displays the probability of the steering angle in the dataset. The histogram shows that small steering angles dominate, but also sharp turns exist.

Analysis of torque and speed: In addition to the steering angle the Udacity dataset also provides measurements of the torque and the speed. These are also potential targets for predictions. Fig. 3.7 shows the steering commands over time for the test set `CH2_001`. The torque strongly correlates with the angle. We also see that the speed is in correlation with the steering angle, *i.e.* a high steering angle is related to a reduction in speed, because otherwise the vehicle would leave the road.

I note that I do not use the torque nor the speed in my experiments in Sect. 3.4, where I focus on improving the steering angle prediction.

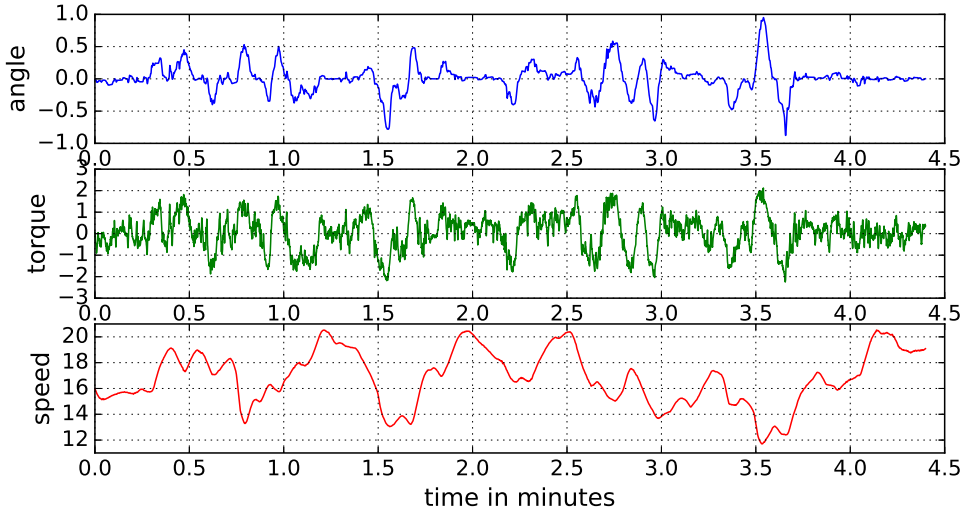


Figure 3.7.: Steering commands of the test set `CH2_001` plotted over time. The steering commands contain the steering angle, the torque, and the speed of the vechicle.

3.3.2. Comma.ai driving dataset

Comma.ai provides a dataset [4] to train self-driving cars. This dataset is used to train the driving simulator of Santana *et al.* [94]. The dataset consists of 10 video clips (45 GB compressed, 80 GB uncompressed) recorded with an Acura ILX 2016 driving in North America. The dataset contains 7.25 hours of driving mostly on highways using one camera mounted behind the windshield. Additionally, measurements of different sensors are stored with the respecting time stamps. These measurements include for example the speed, acceleration, and the steering angle.

3.3.3. Princeton dataset

The Princeton dataset [10] is a virtual dataset, and is also suitable for learning steering command predictions and it contains a training set for the racing simulator TORCS with the vision of the driver (comparable to the center camera) and log data with the ego-motion of the car.
Overall the dataset consists of 12 hours footage (50 GB) of a human player driving on the tracks and it is stored in the leveldb-format.

Fig. 3.8 shows the driver's vision in TORCS and a visualization of the position prediction of the cars [10].



Figure 3.8.: Example image of TORCS from [10]. The image shows the view of the driver and on the side the visualization of the predicted positions of the cars.

3.3.4. Berkley DeepDrive Video dataset

The Berkley DeepDrive Video (BBD-V) dataset [124] is recorded using multiple cars equipped with dash-cams. It uses a crowd-source approach, where dash-cam video clips have been recorded in multiple US-American cities and in different driving conditions (night and day). The driving scenes show various scenarios (city traffic, highways, rural zones).

All video clips are available in high definition (1280×720 pixels) and with sensor measurements from GPS and IMU. Additionally, the dataset contains vectors describing the ego motion of the car (*i.e.* direction and speed). The sampling frequency of the dataset is 15 Hz and the video clips are approximately 38 seconds long. The dataset already split in a training set (28641 video sequences – 79.3%), validation set (2500 video sequences – 6.9%) and test set (5000 video sequences – 13.8%).

This dataset contains video material of approximately 10,000 hours. Fig. 3.9 shows some example video frames of the dataset. Notably, there are various driving scenes at night time where the surroundings are very dark. Occasionally, some videos only show how the car is standing in the street without anything moving.



Figure 3.9.: Sample images from the BDD-V dataset [124].

Analysis of the BDD-V dataset

I analyze the ground truth of the BDD-V training set to evaluate how the steering commands are distributed. The evaluation is based on the processed ground truth for the BDD models [124] (see Sect. 3.4.3). Fig. 3.10 shows the distribution of the four steering commands ('straight', 'stop', 'left', and 'right') for the discrete BDD models. We notice the high probability for driving straight or stopping, whereas the probability for left and right turns is low. Evaluating the discretized steering angles for the continuous BDD model shows the same results. Fig. 3.11 illustrates that there many driving scenes for driving straight, whereas there is low probability of turns. Notably, the distributions are the same for the validation set.

This evaluation shows that the dataset is very unbalanced. For learning lane following, pre-processing would be necessary to reduce the strong appearance of driving straight.

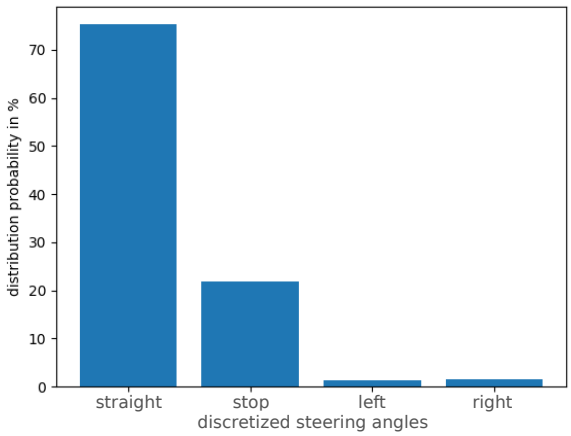
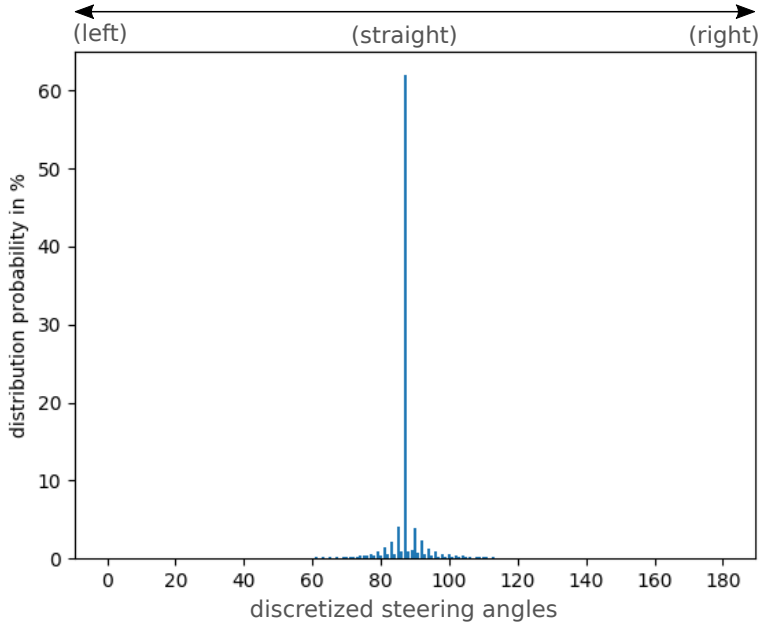
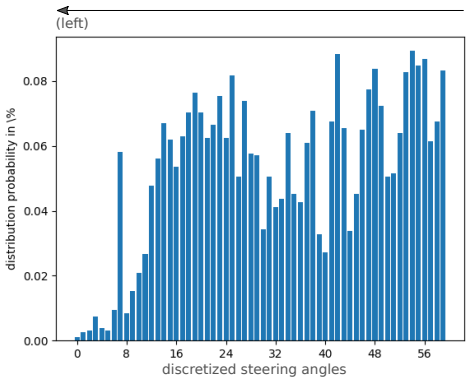


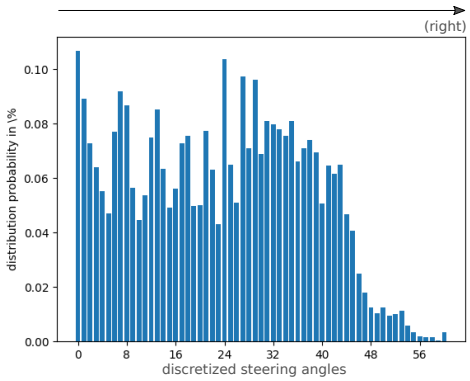
Figure 3.10.: Steering command distribution in the BDD-V dataset [124]. The distribution shows the probability of the ground truth bins of the discrete BDD models with output units for 'straight', 'stop', 'left', and 'right'. We see that the frames for 'straight' dominate, whereas there are not many turns in the dataset. This distribution is computed over the BDD-V training set.



(a) Steering angles from 0 (left) to 180 (right).



(b) Snippet of (a) showing angles between 0 and 69. We see that left turns occur in the dataset, but they are rare.



(c) Snippet of (a) showing angles between 120 and 179. We see that right turns occur in the dataset, but they are rare.

Figure 3.11.: Steering angle distribution of the ground truth in the BDD-V dataset [124], where the steering angle is discretized for the continuous BDD model. This distribution is computed over the BDD-V training set. We notice that the steering angle for driving straight highly dominate in the dataset, whereas turns appear only with a low probability.

3.4. Driving models

First, I review two models with a single frame as input (Sect. 3.4.1) like the Nvidia driving model [6] and ResNet-50 [42]. Next, I analyze the driving models of the winning teams from the Udacity Challenge 2 [9], where teams trained deep driving models on a lane following task (Sect. 3.4.2). Lastly, I present in Sect. 3.4.3 the Berkley DeepDrive (BDD) models [124], which are used for the visualization experiments in Chapter 5.

The single-frame models and the Udacity models are trained on the Udacity dataset [110] from Sect. 3.3.1. All of them have a linear output layer and the predictions are optimized using regression. The BDD models are trained on the BDD-V dataset [124] from Sect. 3.3.4 and the driving task is formulated as a classification problem, where the output layer predicts the probability of a particular ego-motion by optimizing the cross-entropy loss.

3.4.1. Single-frame regression models

In this section we describe two single-frame models, *i.e*. they predict the steering commands conditioned on a single image frame as input. The models consist of convolutional and fully-connected layers and do not learn temporal coherences in the driving task.

In Sect. 4.3.1 I train both models on the Udacity dataset for the self-driving task with a linear output neuron. I also generate preferred input images for the models in Sect. 5.3.6.

Nvidia model

Bojarski *et al*. [6] proposed a deep driving model for imitation learning of steering angles. The model consists of a normalization, five convolutional and four fully-connected layers. The models learns to control the steering commands of the vehicle in end-to-end fashion, *i.e*. there is no manual feature selection. Fig. 3.12 shows the architecture of the model.

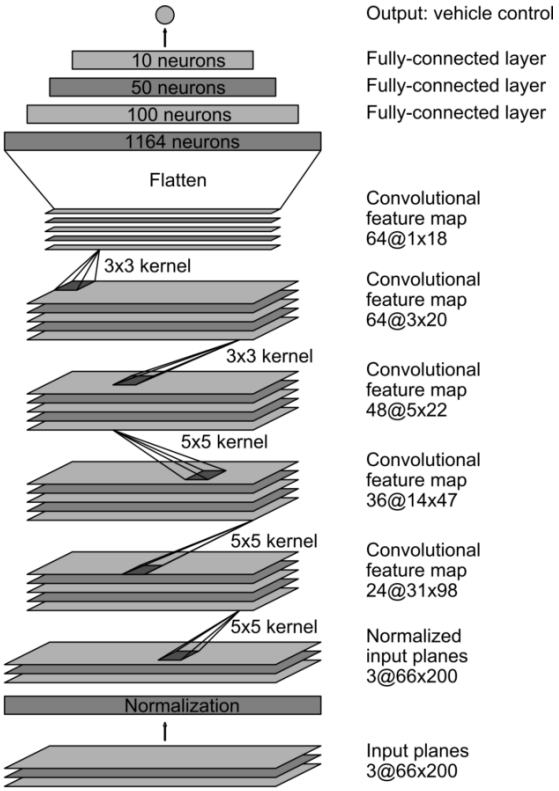


Figure 3.12.: Architecture of the Nvidia model from [6].

ResNet-50

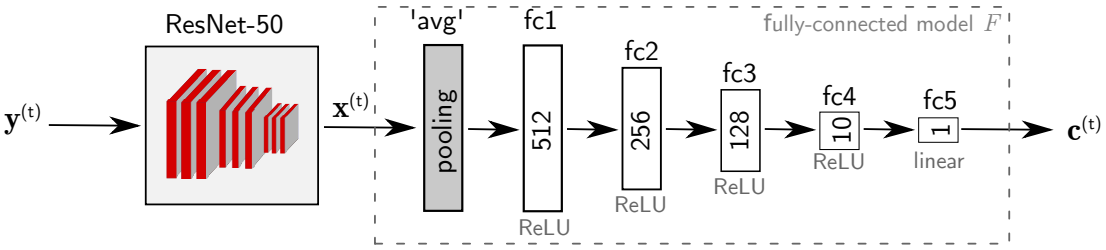


Figure 3.13.: Architecture of the driving model using the ResNet-50 for feature extraction. The dimensionality of the last convolutional layer is reduced with a global average pooling layer.

We use the very deep convolutional network ResNet-50 [42] with model parameters trained on the ImageNet datset [90]. We take the convolutional layers of this trained

model and add fully-connected layers on top (as shown in Fig. 3.13) which reduce the feature dimensionality to the one of the steering commands. The fully-connected layers are initialized randomly. During the training process, the model parameters of the ResNet-50 are fine-tuned on the driving task.

3.4.2. Udacity challenge models

Three of the top-5 models of Udacity's **Challenge 2 "Using Deep Learning to Predict Steering Angles"** use a CNN+LSTM combination, whereas the other two models only use CNN components. A short overview can be found in Table 3.3. These models[2] are open-source and also the code for testing them on a dataset is available. Following I describe the three wining approaches more detailed.

Table 3.3.: Overview of the five winning models of the Udacity Challenge #2 [9]

Team name	Description
Komanda	spatiotemporal CNN + LSTM
Rambo	3-stream model of CNNs inspired by Nvidia [6] and Comma.ai [94]
Chauffeur	CNN inspired by Nvidia [6] + LSTM
Autumn	CNN inspired by Nvidia [6]
cg23	CNN inspired by Nvidia [6] + LSTM

Komanda

Using the input frames of the center camera, this model predicts the steering angle and also the current speed and the torque. The base of the model is a spatiotemporal CNN, which convolves the input video in space and time. On top there is an LSTM network that predicts steering commands conditioned on the feature vector of the CNN and the past predictions.

The CNN consists of four convolutional layers with dropout and four fully connected layers. Residual connections are used as shortcuts after each convolutional layer. An exponential Linear Units (ELU) [18] is used to normalize the output layer of the CNN. There are two modes available which control the input of the recurrent model: The model can learn with the ground truth, *i.e.* the measured steering commands are fed into the next time step. Otherwise the model uses the predicted values for the predictions of the next time step; this is called autoregressive mode.

The loss for training is defined by two error rates: the mean squared error between predicted and true steering angle, and the mean squared error between predicted and true

[2] Available at: `https://github.com/udacity/self-driving-car/tree/master/steering-models`

steering commands (that includes speed and torque for ground truth and autoregressive mode). The trade-off weight on the steering error is higher than the other one, because the evaluation metric of the challenge only takes the steering angle into account.

The model is evaluated after each epoch and the best model – validated on the mean squared error of the steering angle – is stored. The total number of training epochs is 100 and an aggressive regularization via dropout is used.
Testing happens each time when the validation score decreases and the predictions are written in files for later evaluation.

This approach uses TensorFlow-Slim [40], a high-level library for simple model definitions and exploration.

Rambo

This approach only uses CNNs without LSTM layers. The code contains 13 models with different numbers of neurons and layers (10 inspired by Nvidia [6], 3 inspired by comma.ai [94]). However, the final model is build of three streams which are merged at the final layer which is shown in Fig. 3.14. The input to the model is a difference image of two frames in gray-scale.



Figure 3.14.: Model overview of team Rambo from [85]. It shows the three streams of convolutional and fully connected layers which are merged to a single prediction - the steering angle conditioned a difference image in gray-scale.

This approach provides two scripts for preprocessing the data (training data and test data). The images of the center camera are processed and then stored together with the steering angles in arrays.

The model is implemented in Keras [17]. The code provides training and testing scripts and a visualization, which I extend to visualize my models. Furthermore, a script in the language R is provided. This script shows statistics about the results and some examples of images with the respecting steering angles (true and predicted).

Chauffeur

This team uses a CNN with inception architecture [72] and an LSTM on top. They trained the CNN on the Udacity dataset with a fully-connected layer to learn spatial feature extraction. The extracted features were used to train the LSTM model for temporal coherence. Fig. 3.15 shows the architecture of the CNN. The CNN is pretrained on the driving task using one full-connected layer (which is not used in the final model). The LSTM model on top of the convolutional features consists of a recurrent layer with 256 neurons and two fully-connected layers which reduce the dimension to 1 – the regressive steering angle. Early stopping serves as regularization method for a maximum of 100 epochs.



Figure 3.15.: Overview of the CNN of team Chauffeur.

3.4.3. Berkley DeepDrive models

For the neural network visualizations in Chapter 5 we use the Berkley DeepDrive (BDD) models [124] which are trained on the BDD-V dataset, Sect. 3.3.4. All models treat the driving task as a classification problem which provides the opportunity to train models on multimodal data, *i.e.* the model can learn to understand for example driving scenes with crossings where the prediction of multiple directions is valid.

Used BDD models:

- **Discrete models:** These models are trained with four output neurons for the driving commands: “straight”, “stop”, “left”, and “right”. The output layer uses the softmax function to compute the probability of each driving command.
  - *CNN:* A CNN with a single frame as input and fully-connected layers on top of the convolutional features. The architecture is illustrated in Fig. 3.17a.
  - *CNN-LSTM:* A CNN with spatial filters and multiple input frames. The sequence of convolutional features is processed with an LSTM. I show the architecture in Fig. 3.17b.
  - *FCN-LSTM:* An LSTM on top of a fully convolutional network (FCN), *i.e.* larger feature maps at the outputs, with dilated filters processing multiple input frames. The architecture of this model is shown in Fig. 3.18a.
- **Continuous model:** This model has 360 datadriven discretized output bins, where the first 180 bins are used to classify the direction and the other 180 bins correspond to speed predictions (see Fig. 3.16). The output layer uses the softmax function to compute the probability over all steering commands. With this method it is also possible to predict steering angles at a crossing, where the car could go into multiple directions. The architecture is shown in Fig. 3.18b.

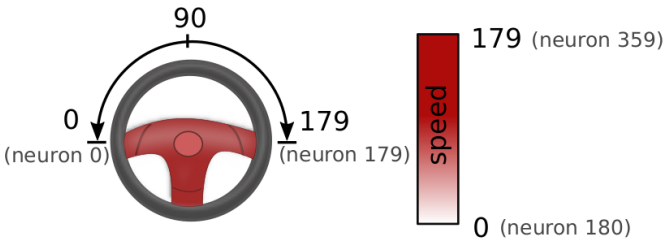
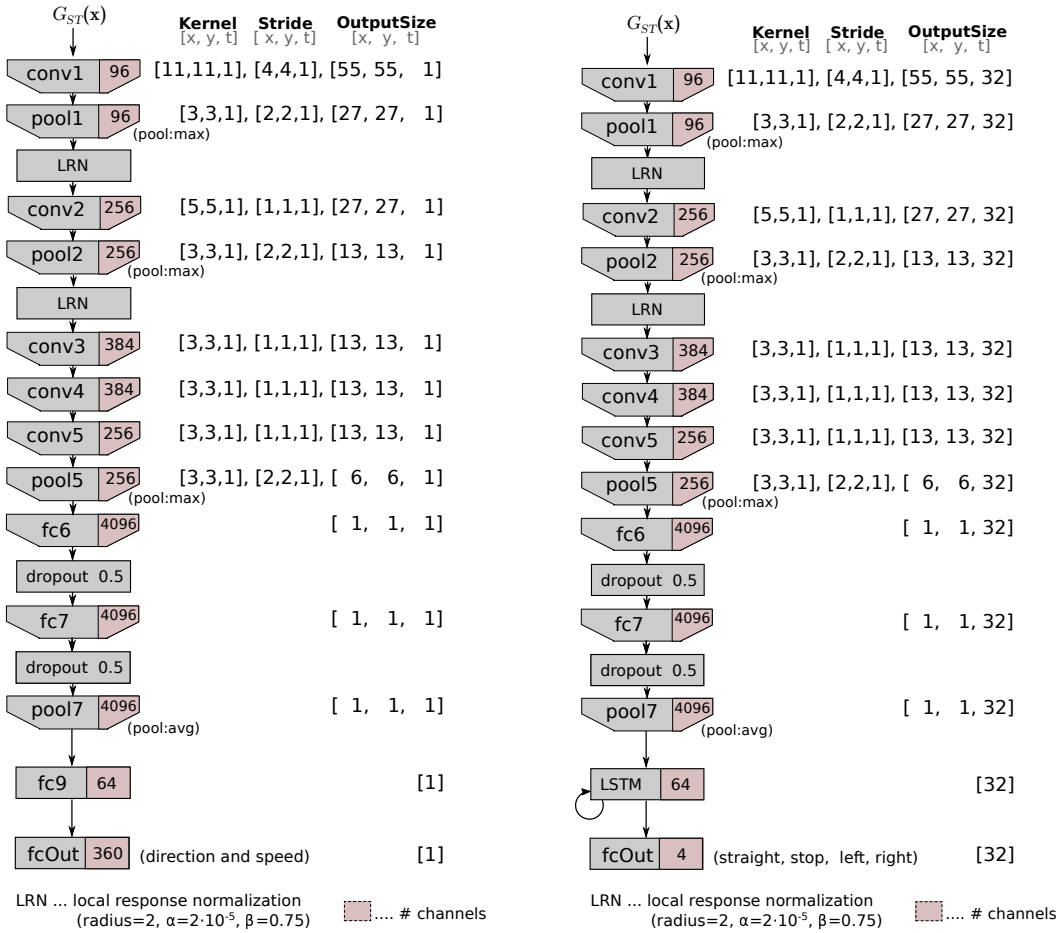
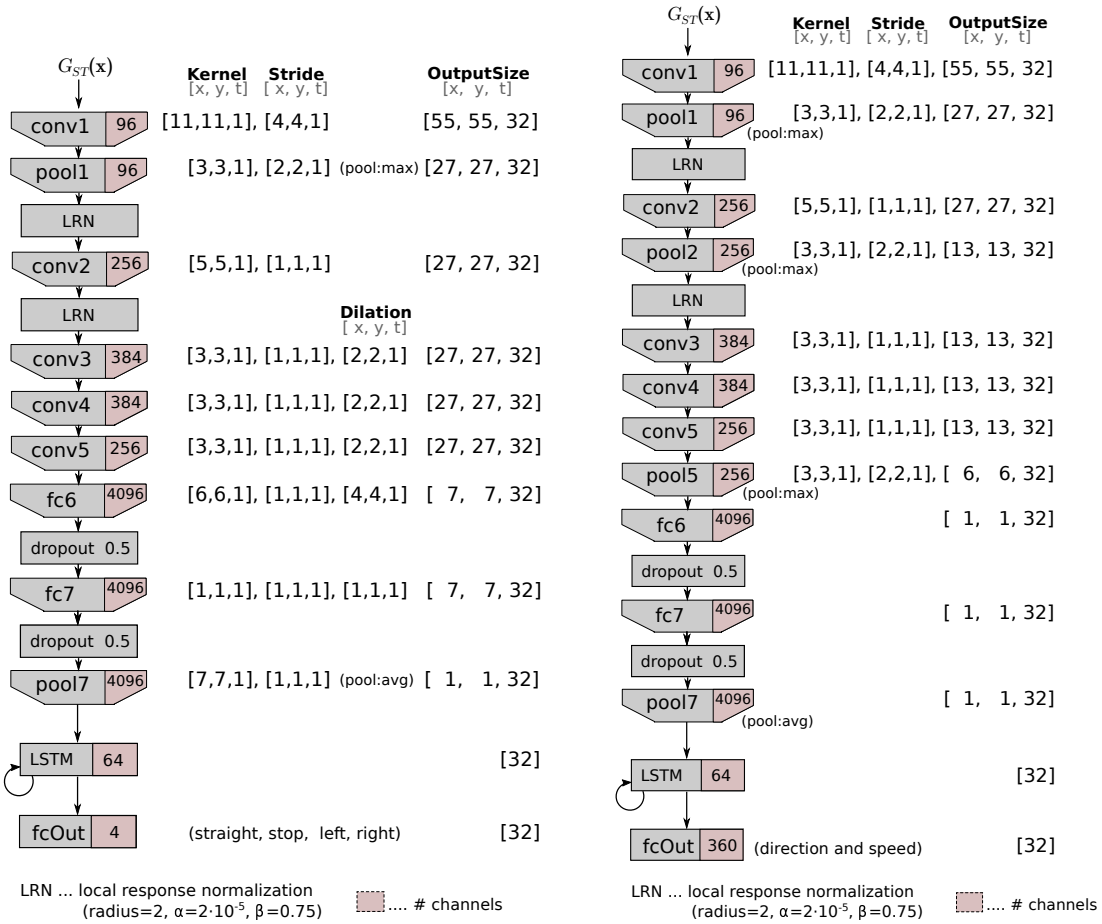


Figure 3.16.: Output neurons of the continuous BDD model which are divided into steering and speed neurons.



(a) CNN model with single-frame input (b) CNN-LSTM model

Figure 3.17.: Architectures of the driving models [124]: CNN (with single frame input) and CNN-LSTM. In the right column I show the geometric properties at each individual layer for a $227 \times 227 \times 32$ sized video input.



(a) FCN-LSTM model (b) Continuous BDD model

Figure 3.18.: Architectures of the driving models [124]: FCN-LSTM and the continuous BDD model. In the right column I show the geometric properties at each individual layer for a $227 \times 227 \times 32$ sized video input.

4. Analyzing & Improving Driving Models

After showing that an end-to-end trained CNN with some fully-connected layers on top can predict the steering angles of an autonomous car [6], in the Udacity Challenge 2 many teams worked on improving driving models for a lane following task. Training the models end-to-end means that no manual feature selection is necessary. However, to make the training easier, the team Chauffeur first trained the CNN on the driving task to learn spatial structure in driving data. Afterwards, the LSTM was trained to learn the temporal structure. For my experiments I use the same dataset and the same evaluation metric as in the challenge to be able to compare the performance of my model to the performance of the winning models.

This chapter describes experiments with models that predict future steering commands, *i.e*. the steering commands for the current and the next time steps. The goal is to use the future predictions of previous time steps to improve the steering prediction at the current time step.

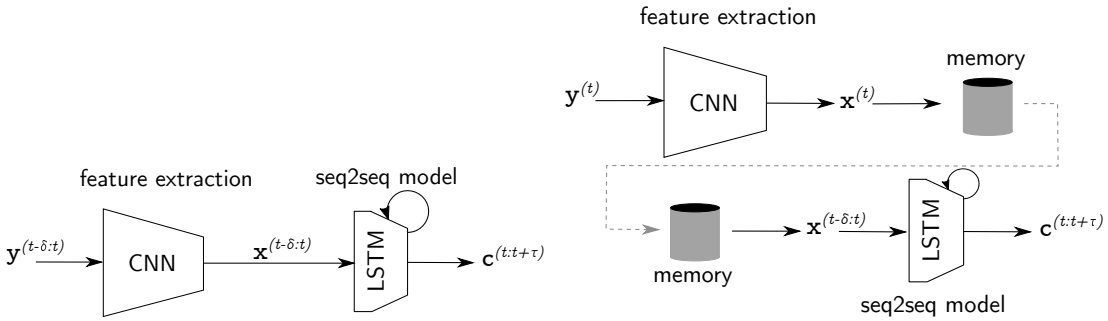
All my models consist of a CNN and a sequence-to-sequence (seq2seq) model, which are trained either end-to-end or separately (Sect. 4.1.1). In Sect. 4.1.3 I propose a future prediction model, that is conditioned on the past and predicts steering angles for the current and future time steps. Sect. 4.1.4 shows the architectures of the seq2seq models and Sect. 4.1.2 describes the used CNN architectures. Sect. 4.1.5 describes the method for superimposing the current steering angle with the future predictions of the previous time steps.

In the experimental setup (Sect. 4.2) I describe the implementation details for the baselines and the seq2seq models, the used data augmentation to prevent overfitting, the hyperparameters and the evaluation metric. Sect. 4.3 shows the experimental results.

4.1. Methods

4.1.1. Architecture overview

Driving models can be trained end-to-end, that means the parameters of the CNN and the seq2seq model are trained simultaneously as shown in Fig. 4.1a. This has the advantage that the two models can align perfectly to each other. The drawback is that training the model takes a long time when using deep CNNs like ResNet-50. One possible solution is to extract all feature vectors, $\mathbf{x}^{(t)} = CNN(\mathbf{y}^{(t)})$, and store them in memory as shown in Fig. 4.1b. When training the seq2seq model, the sequences of feature vectors, $\mathbf{x}^{(t-\delta:t)}$, with δ frames can be loaded from memory and the duration of the training procedure drops. The lower training duration is an advantage for testing models with different architectures or parameters, but for bigger dataset or heavy augmentation it is not feasible to store all extracted feature vectors.



(a) Architecture 1: End-to-end training of the CNN and the seq2seq model. This method can be used for learning both networks from scratch or for training the the seq2seq model while fine-tuning the CNN.

(b) Architecture 2: Separate training, where first the CNN extracts the feature vector $\mathbf{x}^{(t)}$ for each frame $\mathbf{y}^{(t)}$ and stores them in the memory. Then, the seq2seq model trains on sequences of feature vectors $\mathbf{x}^{(t-\delta:t)}$.

Figure 4.1.: Architecture overview, where δ denotes the number of frames in the input sequence, τ denotes the number of future predictions, and t indicates the current time step.

4.1.2. CNN architecture

Each model consists of a CNN for feature extraction and on top either fully-connected layers or a seq2seq model which map to the steering command. The input of the CNN is an RGB video $\mathbf{y}^{(t-\delta:t)}$ and it outputs a sequence of feature vectors $\mathbf{x}^{(t-\delta:t)}$. The CNN can be trained in end-to-end fashion together with the seq2seq model or independently with fully-connected layers on top of the convolutional layers. In the latter case, the fully-connected layers are discarded after the CNN-training and not used in the final model.

I use following CNN architectures:

- **CNN with spatial filters:** All video frames are processed by the same two-dimensional CNN filters. That means, the CNN cannot learn any temporal coherences.
  - *Chauffeur CNN*: This CNN has 18 convolutional layers (including two inception blocks). The CNN also uses batch normalization and max pooling layers. We use the model parameters of the Chauffeur team, which already trained it on the driving task using one fully-connected layer (see Sect. 3.4.2).
  - *ResNet-50*: A deep residual CNN [42] pretrained on ImageNet. An average pooling layer reduces the dimensionality of the feature vector after the last convolutional block.
  - *Nvidia CNN:* A relatively small CNN with a normalization layer and five convolutional layers [6]. The architecture of this CNN with the fully-connected layers on top is showed in Fig. 3.12 .
- **Spatiotemporal CNN:** It convolves the video in space and time, *i.e*. it learns also temporal coherences.
  - *Komanda CNN:* a three-dimensional CNN with residual connections (described in Sect. 3.4.2).

4.1.3. Future prediction approach

In the future prediction approach I do not only predict the current steering commands but also the steering commands for future time steps, *i.e*. the model predicts into the future.
The overview of the approach is shown in Fig. 4.2. The video frames are first processed by a CNN. The seq2seq model uses a sequence of the computed features to predict steering commands for the current and future time steps conditioned on the current and past frames. The recurrent layers in the seq2seq model enable the learning of temporal structure in the driving task.

The Udacity dataset provides video frames from three front-facing cameras as input for the artificial neural network. Principally, only the video frames of the center camera are necessary to train the model. However, the video frames of the left and right camera can be used to enlarge the dataset via data augmentation (more details in Sect. 4.2.2).

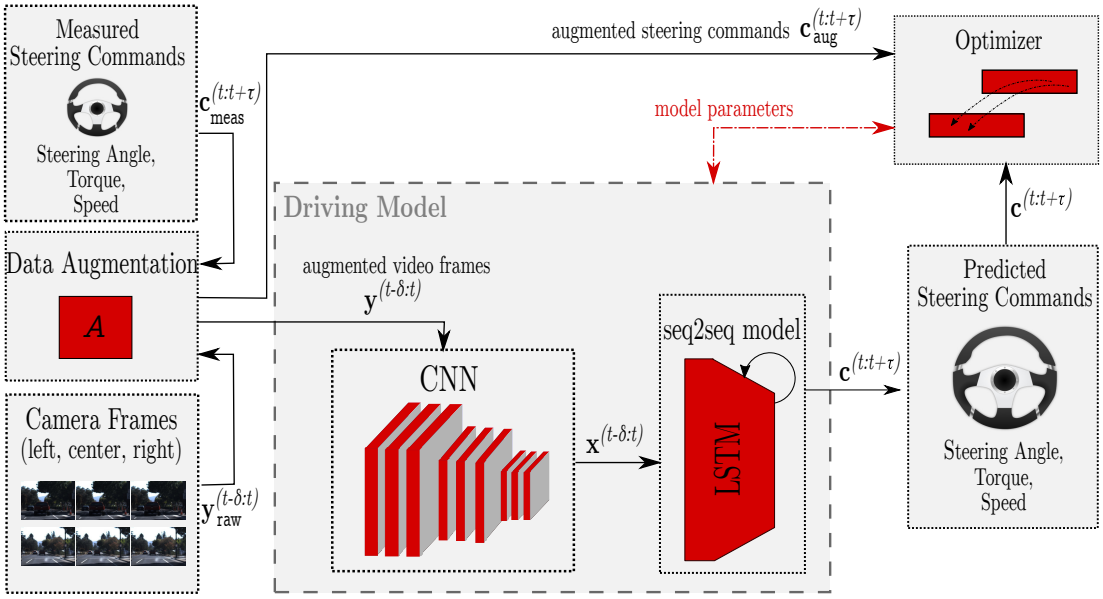


Figure 4.2.: Overview of the seq2seq approach, where the augmented video frames, $\mathbf{y}^{(t-\delta:t)}$, are fed into a CNN which extracts the features $\mathbf{x}^{(t-\delta:t)}$. From the sequence of feature vectors the seq2seq model predicts the current and future driving commands $\mathbf{c}^{(t:t+\tau)}$. The optimizer compares the augmented measured steering commands, $\mathbf{c}_{\text{aug}}^{(t:t+\tau)}$, with the predicted steering commands, $\mathbf{c}^{(t:t+\tau)}$, where δ denotes the number of frames in the input sequence, τ denotes the number of future predictions, and t indicates the current time step.

Sequence-to-sequence model

The CNN extracts the feature vectors $\mathbf{x}^{(t-\delta:t)}$, which are the input data for the seq2seq model. Conditioned on this video sequence the seq2seq model predicts the steering commands $\mathbf{c}^{(t:t+\tau)}$:

$$\mathbf{c}^{(t:t+\tau)} = \text{LSTM}\Big(\text{CNN}\Big(\mathbf{y}^{(t-\delta:t)}\Big)\Big) \tag{4.1}$$

where t is the current time step, and $\mathbf{y}^{(t-\delta:t)} = A(\mathbf{y}_{\text{raw}}^{(t-\delta:t)})$ is the augmented video sequence. The parameter τ denotes the number of predictions into the future, whereas δ denotes the number of frames in the video sequence. Video frames from the future cannot be used, because that would break causality. However, it is possible to use future steering commands for supervised training.

The values of the steering commands are continuous; therefore, their prediction is a regression task and the $\mathcal{L}_2$ loss is used:

$$\mathcal{L}_2 = \sum_{i=t}^{t+\tau} ||A(\mathbf{c}_{\text{meas}}^{(i)}) - \mathbf{c}^{(i)}||_2^2, \tag{4.2}$$

with i indexing over all predictions at the time step t and $A(\mathbf{c}_{\text{meas}}^{(i)})$ being the augmented

measured steering commands, *i.e.* the ground truth. At each time step τ future predictions are generated. That means the loss is computed and summed up for predictions of at the time step t until predictions of time step $t + \tau$.

Alternatively one could also bin the steering commands to discrete values, which was done for the Berkley DeepDrive (BDD) model [124]. That changes the task to a classification problem. Nevertheless, I did not use that for my experiments, because it would not have been comparable anymore with the Udacity approaches.

4.1.4. Seq2seq architecture

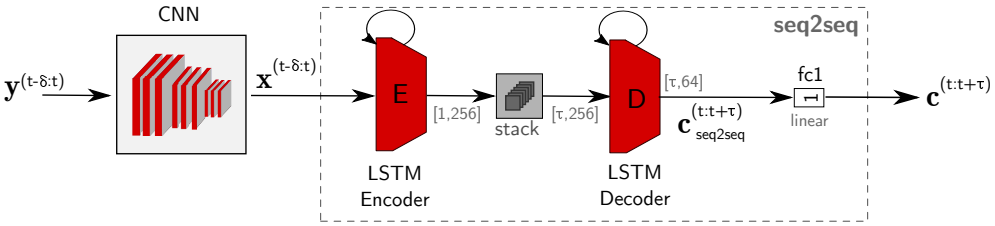
In Fig. 4.3 I propose three different seq2seq models. A simple model Φ_{simple}, a model Φ_{pro} with fully-connected layers for preprocessing the convolutional features and post-processing the output of the LSTM Decoder, and a model Φ_{E2D} with two stacked LSTM encoders. For all architectures a CNN first extracts the feature vectors $\mathbf{x}^{(t-\delta:t)}$ from the input sequence $\mathbf{y}^{(t-\delta:t)}$. These feature vectors are used as input sequence for the seq2seq model, which predicts the future predictions of the steering commands $\mathbf{c}^{(t:t+\tau)}$.

The simple model Φ_{simple} (Fig. 4.3a) consist of an LSTM encoder with 256 neurons and an LSTM decoder with 64 neurons. A fully-connected layer with one output neuron for each future prediction and a linear activation function, serves as output for the regression task, *e.g.* a model with $\tau = 10$ has an output layer with 10 neurons. The output layer has the shape $(1 \times \tau)$, with τ being the temporal dimension.

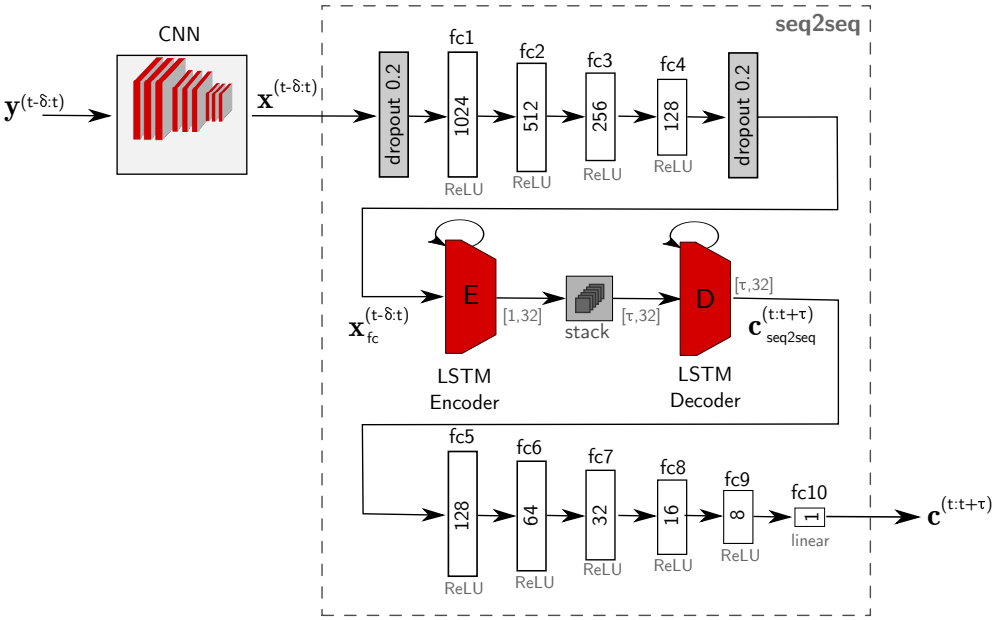
The seq2seq model Φ_{pro} (Fig. 4.3b) first applies dropout on the feature vector $\mathbf{x}^{(t-\delta:t)}$. Four fully-connected layers reduce the dimensionality of $\mathbf{x}^{(t-\delta:t)}$ to the shape $(128 \times \delta)$, and another time dropout is applied. An LSTM encoder-decoder architecture (with 32 neurons in each LSTM) outputs the vector $\mathbf{c}^{(t:t+\tau)}_{\text{seq2seq}}$ conditioned on the processed feature vector $\mathbf{x}^{(t-\delta:t)}_{\text{fc}}$. The output of the decoder is processed by six fully-connected layers. The final layer is the linear output layer with one regressive neuron for each future prediction.

The third model, Φ_{E2D} (Fig. 4.3c), consists of a two-level LSTM encoder (64 neurons for level 1, and 32 neurons for level 2) and a single LSTM decoder (with 32 neurons). The output of the decoder is processed by a fully-connected layer with 10 neurons. Again, the final linear layer outputs the sequence of regressive future predictions $\mathbf{c}^{(t:t+\tau)}$

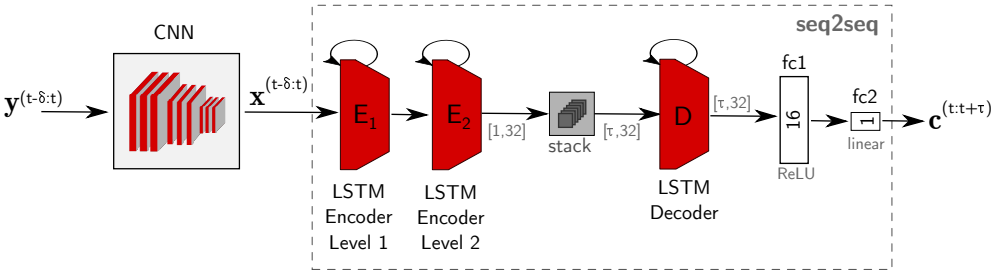
All models are implemented in Keras [17], where it is necessary to implement a 'stack' component between encoder and decoder of the seq2seq model. The encoder processes a sequence of feature vectors and maps them to a vector of shape $[1, n_e]$ with n_e being the number of output neurons of the encoder. To form a vector of the temporal size τ that can be processed by the decoder, the vector of shape $[1, n_e]$ is repeated and stacked in the temporal dimension. This creates a vector with shape $[\tau, n_e]$ that can be fed into the decoder.



(a) Simple seq2seq model Φ_{simple} consisting of an LSTM encoder with 256 output neurons and an LSTM decoder with 64 output neurons.



(b) Seq2seq model, Φ_{pro}, with pre- and postprocessing . The seq2seq model contains an LSTM encoder with 32 output neurons and an LSTM decoder with 32 output neurons. The feature vectors of the CNN, $\mathbf{x}^{(t-\delta:t)}$, are preprocessed using dropout and fully-connected layers. The output of the encoder-decoder model is postprocessed by further fully-connected layers.



(c) Seq2seq model, Φ_{E2D}, with two LSTM encoder levels (level 1: 64 neurons; level2: 32 output neurons) and an LSTM decoder (32 output neurons). Between the decoder and the output layer fc2, a fully-connected layer reduces the data dimensionality further.

Figure 4.3.: Seq2seq model architectures. The component "stack" between encoder and decoder repeats and stacks the encoded vector for τ times to model a sequence, which is then processed by the decoder.

4.1.5. Superposition of future predictions to improve steering

Future predictions of the steering angle can be used for adapting driving commands like speed, *e.g.* to decelerate in front of a turn, but here I want to analyze if the predictions can be used to improve the prediction of the current steering angle with the future range $r = 0$.

We experiment with superpositions of the predictions from the current frame with the future predictions calculated in previous time steps. The superposition principle is shown in Fig. 4.4. The superimposed driving command $\mathbf{c}_{\text{super},i}$ of the current input frame i is the weighted mean of the predictions $\mathbf{c}_{i-r}^{(t+r)}$ where r denotes the future range:

$$\mathbf{c}_{\text{super},i} = \frac{1}{\tau} \sum_{r=0}^{\tau} w_r \mathbf{c}_{i-r}^{(t+r)} \tag{4.3}$$

with i being the index of the frame, t being the current time step, and w_r being an element of the weight vector $\mathbf{w}$ as shown in Fig. 4.4.

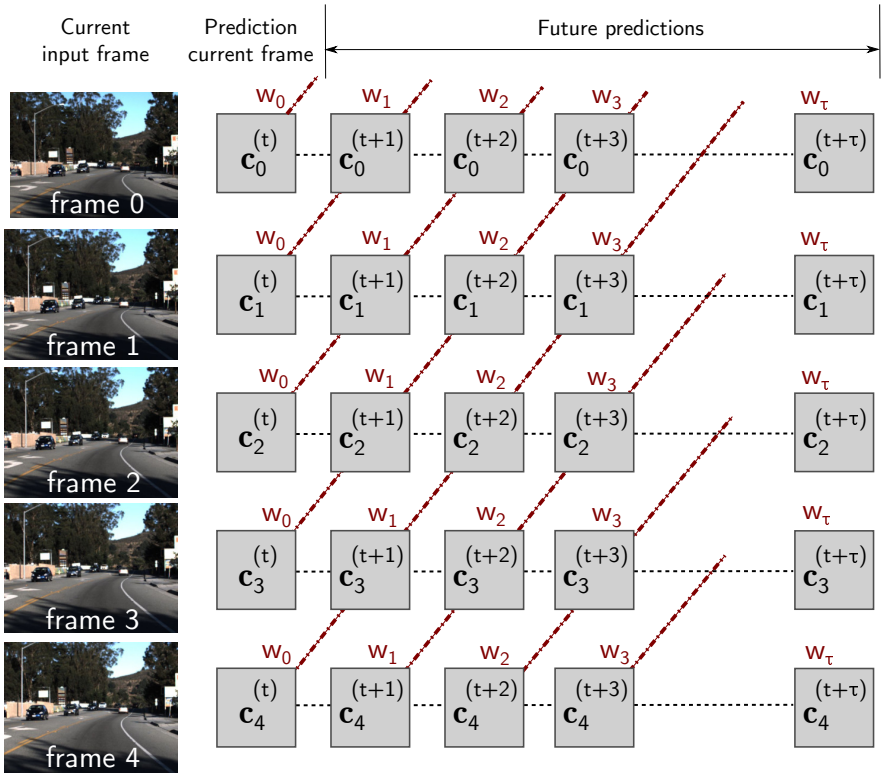


Figure 4.4.: Superposition principle of driving command predictions. The future predictions $\mathbf{c}^{(t:t+\tau)}$ are computed for several time steps, so that previously computed future predictions can be superposed with the prediction of the current time step $\mathbf{c}^{(t)}$. The predictions for the superposition are weighted using the weight vector $\mathbf{w}$, which holds τ weight entries.

The edge case of this method is that no previous future predictions are available. That happens at the beginning of the dataset or in real-world when the autonomous car is activated. For that case either the missing values are assumed to be equal to the last available prediction or the weighted mean is only computed over the available values.

Learning the superposition: Instead of manually setting up a linear combination of weights and driving command predictions, it is also possible to learn a non-linear superposition using a neural network. For this task I use a small feed-forward neural network with two layers: a fully-connected layer with five output neurons and a sigmoid activation function, and another fully-connected layer with one output unit and a linear activation for regression. The model overview is illustrated in Fig. 4.5.

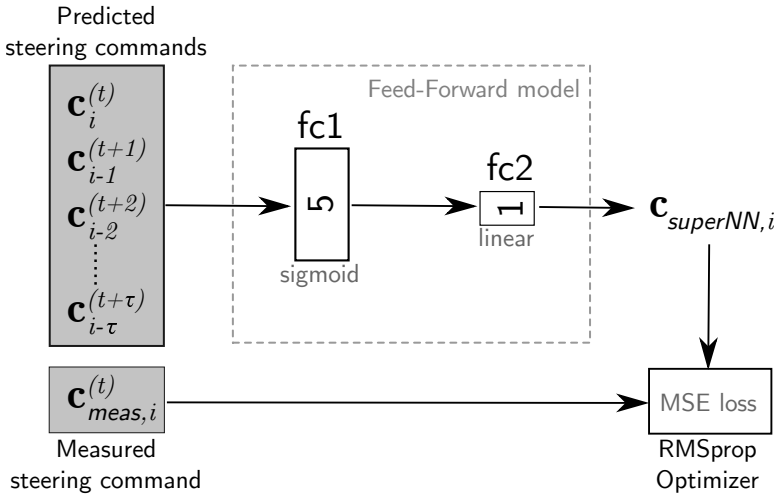


Figure 4.5.: Superposition learned with a feed-forward neural network. The model takes the predicted current steering command, $\mathbf{c}_i^{(t)}$ and its corresponding future predictions from previous sequences, $\mathbf{c}_{i-r}^{(t+r)}, r \in \mathbb{N}_0, r \leq \tau$, as input and outputs the superimposed steering command $\mathbf{c}_{\text{superNN},i}$.

4.2. Experimental setup

For the experiments the Udacity dataset is used. To be able to compare the performance of our model with the approaches of the Udacity Challenge 2, the same training and testing conditions were established. The dataset `CH2_002` (consists of four tracks, total duration 28 minutes) is used for training and validation; the dataset `CH2_001` (with one track, duration 4 minutes 40 seconds) is the testing set. More details about the dataset can be found in Sect. 3.3.1.

This section describes the implementation details, data augmentation, the conducted experiments, and the used hyperparameters for the training procedure of the models.

4.2.1. Implementation details

Seq2seq model: The driving models are all implemented in Keras [17], a high-level machine learning framework for easy model definitions. Keras works on top of Tensorflow [1]. All models are trained on regression with a mean squared error loss. I use the RMSprop optimizer [108] with $\rho = 0.9$ and clipping gradients at a value of 10.

Feed-forward network for learning the superposition: The model, implemented in Keras [17], is trained on the entire Udacity training set `CH2_002` (with 40% of it as validation set) using the RMSprop optimizer [108] and clipping the gradient at value 10. The training lasts for 40 epochs using a learning rate of $2 \cdot 10^{-5}$ and a mini-batch size of 32. The model optimizes the mean squared error (MSE) between the predicted superimposed steering command, $\mathbf{c}_{\text{superNN},i}$, and the measured steering command, $\mathbf{c}^{(t)}_{\text{meas},i}$, from the dataset.

Hyperparameters of the experiments: Table 4.1 lists a detailed description of the used hyperparameters and Table 4.2 lists the values of the hyperparameters for all conducted experiments.

Table 4.1.: Hyperparameter description for the seq2seq model training.

Parameter	Description
δ	the temporal size of the input sequence $\mathbf{y}^{(t-\delta:t)}$; *i.e.* the number of video frames in one input sequence.
Δt	temporal offset of frames in the input sequence: $\mathbf{y}^{(t-(\delta-1)\Delta t, t-(\delta-2)+\Delta t, \ldots, t)}$. Please note that this parameter has been omitted in the other equations for easier reading.
$\mathbf{R}$	future prediction range containing a list of distances r. These distances describe how far a future prediction is looking into the future ($0 \leq r \leq \tau, r \in \mathbb{N}$). All the models have a range $r = 0$, which specifies the prediction for the current frame. It is not necessary that $\mathbf{R}$ contains all possible ranges up to τ, *e.g.* $\mathbf{R} = [0, 5, 10, 20]$ is a valid configuration for $\tau = 20$.
f_s	sampling frequency for the training procedure in fps. The sampling frequency of the Udacity dataset is 20fps.
s_b	batch size for training. For testing the batch size is set to 1.
N_{train}	max. number of training epochs. Early stopping is used to stop the training if the validation loss is not decreasing for longer than 18 epochs.
η	learning rate for the training procedure. The learning rate is reduced by half every time the validation loss plateaus for more than 8 epochs.

Table 4.2.: Hyperparameter settings of the experiments. A description of the hyperparameters can be found in Table 4.1. The row "CNN random-init." states if the CNN is randomly initialized or if trained model parameters are loaded.

ID	**Nvidia**	**ResNet-50**	**S1.1 – S1.12**	**S2.1**	**S2.2**	**S2.3**
CNN	Nvidia	ResNet-50	Chauffeur	ResNet-50	Nvidia	Komanada
CNN random-init.	yes	no	no	no	yes	yes
training CNN	yes	yes	no	yes	yes	yes
augmentation	yes	yes	no	yes	yes	yes
δ	1	1	5	5	5	15
Δt [in frames]			10	10	10	10
f_s [in fps]	10	10	5	5	10	10
s_b	64	16	32	16	8	64
N_{train}	100	100	100	100	100	40
η	10^{-4}	10^{-4}	10^{-4}	10^{-4}	10^{-4}	10^{-4}

4.2.2. Data augmentation

Because of the small size of the dataset, the neural network overfits easily, *i.e.* the network learns the noise in the data and does not generalize anymore. To prevent overfitting we use data augmentation to enlarge the training set artificially.

For video frames there are several ways for augmentation:

- **Augmentation using left/right camera:** Instead of only using the video sequences of the center camera I also train the models on videos from the outer cameras. The steering angle is adjusted according to the direction of the camera.
- **Flipping frame:** The video frame is flipped horizontally and the steering angle is negated. With this augmentation method it is possible to compensate an imbalance between left and right steering angles.
- **Change of image brightness:** Depending on the light conditions, the brightness in images is not always the same. This augmentation method simulates different illuminating by transforming the video frames to the HSV color space. For each frame I multiple the brightness channel V of the transformed image with a number v_{rand} from the uniform distribution $\mathbb{U}(0.4, 1.2)$. The multiplication with $0.4 < v_{\text{rand}} < 1$ makes the image darker and using a value $1 < v_{\text{rand}} < 1.2$ produces a lighter image. After modifying the brightness channel the image is transformed back to the RGB color space.
- **Mask upper image parts:** The networks tend to overfit on the horizon line and the sky in the images. This method masks randomly parts of the upper image half, so the network does not always see the horizon. For an image hight of 240 pixels, I draw a random number h_{rand} from the uniform distribution $\mathbb{U}(0, 120)$. Then I mask all pixels from row 0 (top of the image) to row h_{rand} (maximal the half of the image), *i.e.* I replace the RGB values with zeros.

4.2.3. Evaluation metric

For evaluation the Root Mean Squared Error (RMSE) between the predicted steering command $\mathbf{c}^{(t:t+\tau)}$ and the true steering command $\mathbf{c}_{\text{meas}}^{(t:t+\tau)}$ is used:

$$e_{\text{RMSE}} = \text{RMSE}(\mathbf{c}, \mathbf{c}_{\text{meas}}) = \sqrt{\frac{1}{T}\sum_{i=0}^{T-1} ||\mathbf{c}^{(i)} - A(\mathbf{c}_{\text{meas}}^{(i)})||^2}, \tag{4.4}$$

with T being the number of samples and A being the augmentation function.

For this challenge it has not been necessary to deal with wrong predictions and their consequences. In a real-world situation the car has to learn how to correct driving mistakes, *i.e.*, bad predictions. This behavior cannot be checked with this evaluation metric, because a previous prediction would have led to a different input video frame. To test real driving performance for autonomous driving a simulation would be better suitable to evaluate the model.

4.3. Experiments

In Sect. 4.3 is structure the experiments as following: First, I present the results for the baselines (Sect. 4.3.1), where I also mention the error a model would have if it drives strictly straight ahead. Second, I analyze the reliability of the single future predictions (Sect. 4.3.2), *i.e.* how far can the model look into the future without loosing to much prediction accuracy. Then, I show the results of combining the future predictions from previous time steps to improve the current prediction Sect. 4.3.3. Finally, I present results of experiments with different model architectures (Sect. 4.3.4).

I investigate if future predictions can improve the prediction for the current time step. Therefore, I train multiple models and compare the error for each future range r. Then I analyze if a superposition with the future predictions $\mathbf{c}^{(t:t+\tau)}$ improves the current prediction $\mathbf{c}^{(t)}$.

4.3.1. Baselines without future predictions

As simple baseline I use the architecture of the Nvidia and the ResNet-50 model as shown in Sect. 3.4.1. We map one input frame $\mathbf{y}^{(t)}$ to one steering command $\mathbf{c}^{(t)}$:

$$\mathbf{c}^{(t)} = F\Big(\text{CNN}(\mathbf{y}^{(t)})\Big) \tag{4.5}$$

with $F(.)$ describing the fully-connected model architecture.

The Nvidia model [6] is trained from scratch, *i.e.* all model parameters are initialized randomly. Fig. 3.12 shows the the architecture of the model.

The next experiment uses the convolutional layers of the ResNet-50 model [42] which is pretrained on ImageNet [90]. On top of the convolutional features, an average pooling and four fully-connected layers are added as shown in Fig. 3.13.

The hyperparameter settings for both experiments are listed in Table 4.2 (ID: 'Nvidia' and 'ResNet-50').

The most simple base line for these experiments is to assume a model with a constant steering angle of zero, in other words the model is driving only straight ahead. Using the test set `CH2_001` I get a base line error:

$$e_{\text{RMSE, straight}} = \text{RMSE}(\mathbf{0}, \mathbf{c}_{meas}) = 0.2068 \tag{4.6}$$

over all samples of the test set.

Furthermore, I compare my models to the winning models[1] of the Udacity Challenge 2, which we have described in Sect. 3.4.2. The error rates of these models are shown in Table 4.3. I note that only the teams Rambo and Chauffeur provide trained model parameters to test the results.

Table 4.3.: Leader board of the Udacity Challenge 2.

Team name	e_{RMSE}
Komanda	0.04829
Rambo	0.05788
Chauffeur	0.05816

Before checking the performance of seq2seq model on the driving task, I establish another baseline by implementing two existing model architectures: Nvidia and ResNet-50.

The performance analysis on the Udacity test set `CH2_001` shows an error of

- $e_{\text{RMSE}} = 0.1142$ for the Nvidia model,
- $e_{\text{RMSE}} = 0.0909$ for the fine-tuned ResNet-50 with fully-connected layers on top.

Looking at the predictions of both models in Fig. 4.6 (with the prediction visualized as black point and the ground truth illustrated as white point on the white circle) we see that the predictions jitter; thus, the ride in such an autonomous car would be poor. Furthermore, the video sequences show that both models have problems at a critical sharp left turn in the end of the sequence. The Nvidia model does not perceive the left turn and sends the car straight over the cliff. The ResNet-50 model recognizes the sharp turn, but the resulting steering is too strong and the car would enter the oncoming lane.

[1] Available at: `https://github.com/udacity/self-driving-car/tree/master/challenges/challenge-2`



Figure 4.6.: Baseline single-frame models driving on the test track of the Udacity dataset. The black point on the circle visualizes the predicted steering angle, whereas the white point shows the measured steering angle. We see that the predictions of both models jitter, but we recognize that it is less for the ResNet-50 model. Best viewed in Adobe Reader where videos play on click with configurable speed.

4.3.2. Reliablity of future predictions

For evaluating the reliability of future predictions, I test seq2seq models with different ranges, *i.e.* how far they predict into the future. I investigate how reliable future predictions are and how far the model can see into the future without making a big error. This evaluation shows if future predictions have the potential to improve the current steering command or if they are not reliable at all.

For all models in this experiment I use the trained Chauffeur CNN with frozen model parameters for the feature extraction. The model maps a sequence of input frames $\mathbf{y}^{(t-\delta:t)}$ to a sequence of steering commands $\mathbf{c}^{(t:t+\tau)}$:

$$\mathbf{c}^{(t:t+\tau)} = \Phi\Big(\text{CNN}(\mathbf{y}^{(t-\delta:t)})\Big) \tag{4.7}$$

with Φ denoting the seq2seq model.

The hyperparameters of the experiments are listed in Table 4.2 for the experiment IDs 'S1.1 – S1.12'. The range of the future predictions is stated in the tables with the results (Table 4.4 to 4.6).

For these experiments we use two different seq2seq model architectures which are shown in Fig. 4.3a and 4.3b. The simple seq2seq architecture, Φ_{simple}, is used in the experiments S1.2, S1.6, S1.8; the other experiments use the seq2seq architecture, Φ_{pro}.

In the experiments the prediction range r indicates how many frames the neural network looks into the future. Range $r = 0$ shows the prediction for the current frame. With a sampling rate of 20fps, a prediction range of 20 frames corresponds to one second.

I evaluate models with future predictions up to 5 seconds in the future and cluster them into three categories depending on their maximum range τ in frames:

1. **Near-range future prediction model:** $0 < \tau \leq 20$ (Model S1.1 – S1.5)
2. **Middle-range future prediction model:** $20 < \tau \leq 50$ (Model S1.6. – S1.10)
3. **Long-range future prediction model:** $50 < \tau \leq 100$ (Model S1.11 – S1.12)

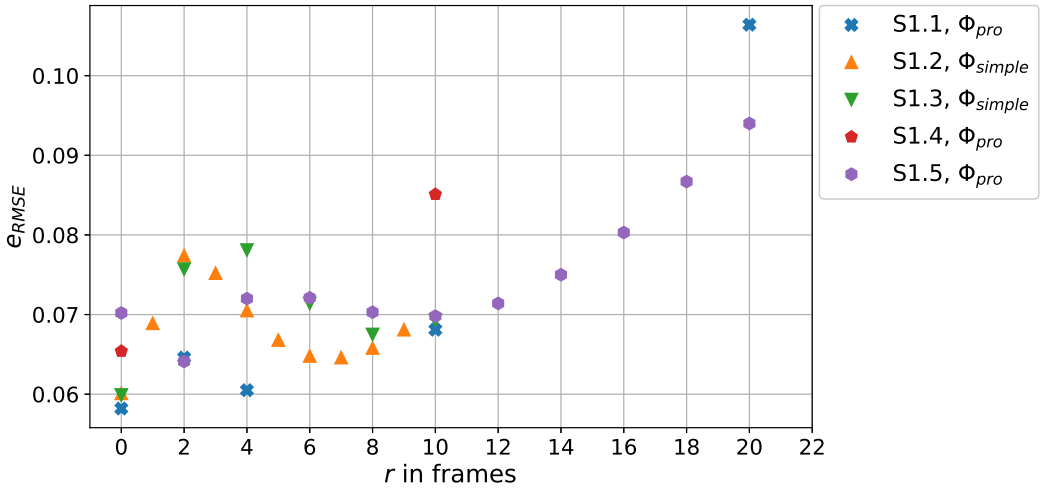


Figure 4.7.: Prediction errors of **near-range** seq2seq models (experiments S1.1 – S1.5). The root mean squared error r_{RMSE} of the current frame is visualized at prediction range $r = 0$.

Table 4.4 shows models with *near-range* future predictions. That means the maximum these models predict into the future is 20 frames. These models show the lowest errors of all models. The errors of the models are plotted in Fig. 4.7. I notice a trend of increasing error e_{RMSE} with increasing range r, but the predictions for $0 \leq r \leq 6$ are rather noisy. The two models, S1.2 and S1.3, with the simple seq2seq module, Φ_{simple}, show both a very low error for the current frame $r = 0$. However, the error rises for future predictions which predict 2 to 4 frames into the future. Models with future ranges below 20 frames, clearly outperform the single-frame models Nvidia and ResNet-50.

Table 4.4.: **Near-range future prediction:** Different seq2seq models using the Chauffeur-CNN for feature extraction. The second row in the table states the used seq2seq architecture (see Fig. 4.3). The prediction range r (in frames) corresponds to the output unit of Φ which describes the temporal distance of the future prediction to the current frame.

S1.1 Φ_{pro}		**S1.2** Φ_{simple}		**S1.3** Φ_{simple}		**S1.4** Φ_{pro}		**S1.5** Φ_{pro}	
r	e_{RMSE}	r	e_{RMSE}	r	e_{RMSE}	r	e_{RMSE}	r	e_{RMSE}
0	0.0582	0	0.0601	0	0.0599	0	0.0654	0	0.0702
2	0.0646	1	0.0689	2	0.0757	10	0.0851	2	0.0641
4	0.0605	2	0.0774	4	0.0781			4	0.0720
10	0.0681	3	0.0752	6	0.0714			6	0.0721
20	0.1064	4	0.0705	8	0.0675			8	0.0703
		5	0.0668					10	0.0698
		6	0.0648					12	0.0714
		7	0.0646					14	0.0750
		8	0.0658					16	0.0803
		9	0.0681					18	0.0867
								20	0.0940

Table 4.5 contains the data of experiments with the *middle-range* models. These models predict a maximum of 50 frames into the future. In Fig. 4.8 a monotonic increase of the errors is visible with bigger prediction ranges r. For models with $r \leq 50$ all future predictions are better than the baseline for driving strictly straight from (4.6). Interestingly, the number of different future prediction ranges has more or less no impact on the quality of the future predictions and there are no real outliers from the visible trend.

Table 4.5.: **Middle-range future prediction:** Different seq2seq models using the Chauffeur-CNN for feature extraction. The second row in the table states the used seq2seq architecture (see Fig. 4.3). The prediction range r (in frames) corresponds to the output unit of Φ which describes the temporal distance of the future prediction to the current frame.

S1.6 Φ_{simple}		**S1.7** Φ_{pro}		**S1.8** Φ_{simple}		**S1.9** Φ_{pro}		**S1.10** Φ_{pro}	
r	e_{RMSE}	r	e_{RMSE}	r	e_{RMSE}	r	e_{RMSE}	r	e_{RMSE}
0	0.0655	0	0.0671	0	0.0696	0	0.0728	0	0.0742
5	0.0749	4	0.0663	5	0.0665	10	0.0802	10	0.0822
10	0.0831	8	0.0726	10	0.0752	20	0.1035	20	0.1072
15	0.0892	12	0.0765	15	0.0816	30	0.1290	30	0.1323
20	0.0999	16	0.0838	20	0.0918	40	0.1566	40	0.1606
25	0.1145	20	0.0938	25	0.1051			50	0.1876
30	0.1311	24	0.1046	30	0.1204				
		28	0.1155	35	0.1366				
		32	0.1268	40	0.1531				
		36	0.1390						
		40	0.1522						

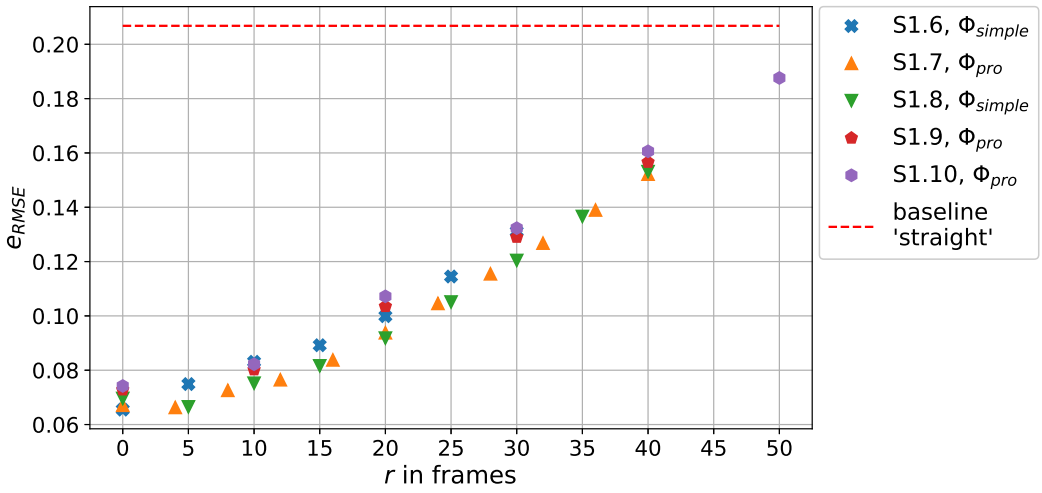


Figure 4.8.: Prediction errors of **middle-range** seq2seq models (experiments S1.6 – S1.10). The root means squared error r_{RMSE} of the current frame is visualized at prediction range $r = 0$. Additionally, the figure shows the baseline error $e_{\text{RMSE,straight}} = 0.2068$ for driving strictly straight (see Sect. 4.3.1).

A third table shows two *long-range* future prediction models. Table 4.6 shows that future predictions with $r \geq 60$ are statistically worse than driving strictly straight ahead. This is also visualized in Fig. 4.9, where we see that predictions with $r \geq 60$ are worse

than the simple baseline. This is reasonable, because for sharp turns, the camera is not able to see the further course of the road. We notice that the error for the current frame $r = 0$ rises the more far the models predicts into the future. However, looking ahead has the advantage that the model could overcome situations where the road is not visible in the moment it approaches the turn, but it has seen the turn already before. This could happen for example because of a glare or a hill which blocks the range of vision.

Next I analyze if the future predictions of previous time steps can be used to improve the current prediction at $r = 0$.

Table 4.6.: **Long-range future prediction:** Different seq2seq models using the Chauffeur-CNN for feature extraction. The second row in the table states the used seq2seq architecture (see Fig. 4.3). The prediction range r (in frames) corresponds to the output unit of Φ which describes the temporal distance of the future prediction to the current frame.

S1.11 Φ_{pro}		**S1.12** Φ_{pro}	
r	e_{RMSE}	r	e_{RMSE}
0	0.0892	0	0.0830
20	0.1123	20	0.1198
40	0.1743	40	0.1785
60	0.2201	60	0.2185
		80	0.2478
		100	0.2755

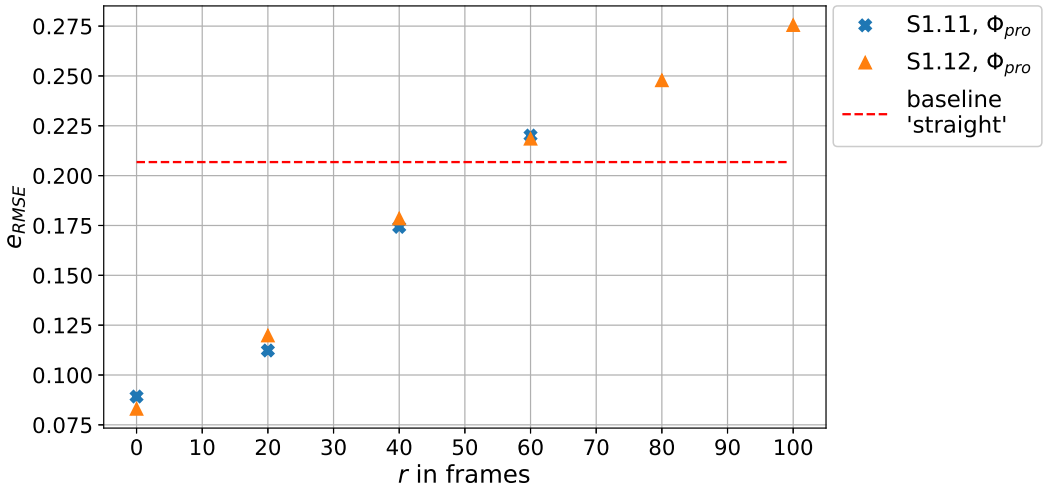


Figure 4.9.: Prediction errors of **long-range** seq2seq models (experiments S1.11 – S1.12). The root means squared error r_{RMSE} of the current frame is visualized at prediction range $r = 0$. Additionally, the figure shows the baseline error $e_{\text{RMSE,straight}} = 0.2068$ for driving strictly straight (see Sect. 4.3.1).

4.3.3. Improving the prediction by combining with future predictions from previous time steps

After analyzing the reliability of future predictions, I analyze if a superposition (as described in Sect. 4.1.5) improves the driving command for the current frame. The models used for this experiments are the ones from the previous section (S1.1 – S1.12).

Using all future predictions: A simple way of combining is to set all weights from equation (4.3) to $w_r = 1, \forall r \in \mathbb{N}, r \leq \tau$. The results of the superimposed steering commands with all future predictions is shown in Table 4.7.

For the near-range models we see that the superimposed error $e_{\text{RMSE,super}}$ changes the result only marginally compared to the error of the current frame ($r = 0$). The middle-range and the long-range models show higher errors for the superimposed error throughout all experiments. This is to be expected, because the more the model looks ahead into the future the less reliable are the future predictions. A simple superposition with all weights equal to 1 does not improve the steering prediction for the current frame.

Table 4.7.: RMSE of the superimposed steering angle $\mathbf{c}_{super}$ using **all future predictions** from previous time steps. In this table I also show the error $e_{\text{RMSE}}(r = 0)$ of the current frame from the previous section for comparison. The colored entries mark the models with improved performance, *i.e.* with lower error.

Near-range	**S1.1**	**S1.2**	**S1.3**	**S1.4**	**S1.5**
$e_{\text{RMSE}}(r = 0)$	0.0582	0.0601	0.0599	0.0654	0.0702
$e_{\text{RMSE,super}}$	0.0587	0.0608	0.0616	0.0625	0.0622
Middle-range	**S1.6**	**S1.7**	**S1.8**	**S1.9**	**S1.10**
$e_{\text{RMSE}}(r = 0)$	0.0655	0.0671	0.0696	0.0728	0.0742
$e_{\text{RMSE,super}}$	0.0726	0.0825	0.0787	0.0857	0.0980
Long-range	**S1.11**	**S1.12**			
$e_{\text{RMSE}}(r = 0)$	0.0892	0.0830			
$e_{\text{RMSE,super}}$	0.1188	0.1360			

Using the most reliable future predictions: As the superposition of all predictions with equal weights is underperforming as expected, I conduct a simple feasibility analysis if it is possible to improve the prediction for the current frame by adaptively scaling the weights. For that I set the weights depending on the reliability of the future predictions from the previous section. I manually select the future predictions with the best errors e_{RMSE} from Table 4.4 , 4.5, and 4.6 and set their weights $w_r = 1$. For all

other future predictions the weights are set to zero. As this method is working on the results of the test set (and not on the training set), it is an oracle that the results improve. Table 4.8 shows the selected future ranges and the resulting superposition errors. Please note that I always include the predictions for the current frame ($r = 0$) and I select a maximum of five prediction ranges r where the weight is set to $w_r = 1$.

We see in Table 4.8 that for all models the superimposed error, $e_{\text{RMSE,super}}$, is lower than the original error at the current frame, $e_{\text{RMSE}}(r = 0)$. Especially the middle- and the long-range models show big improvement, which is reasonable, because the big error for predictions that look far ahead has distorted the performance when using a superposition of all future predictions.

Table 4.8.: RMSE of the superimposed steering angle $\mathbf{c}_{super}$ using **the most reliable future predictions** from previous time steps depending on their reliability. In this table I also show the error $e_{\text{RMSE}}(r = 0)$ of the current frame from the previous section for comparison. The ranges R (in frames) state which predictions I use for the superposition. The colored entries mark the models with improved performance, *i.e.* with lower error.

Near-range	**S1.1**	**S1.2**	**S1.3**	**S1.4**	**S1.5**
$e_{\text{RMSE}}(r = 0)$	0.0582	0.0601	0.0599	0.0654	0.0702
Ranges R	[0,2,4,10]	[0,1,6,7,8]	[0,8]	[0,10]	[0,2,10]
$e_{\text{RMSE,super}}$	0.0560	0.0562	0.0557	0.0625	0.0559
Middle-range	**S1.6**	**S1.7**	**S1.8**	**S1.9**	**S1.10**
$e_{\text{RMSE}}(r = 0)$	0.0655	0.0671	0.0696	0.0728	0.0742
Ranges R	[0,5]	[0,4,8,12]	[0,5,10,15,20]	[0,10]	[0,10]
$e_{\text{RMSE,super}}$	0.0611	0.0604	0.0576	0.0657	0.0662
Long-range	**S1.11**	**S1.12**			
$e_{\text{RMSE}}(r = 0)$	0.0892	0.0830			
Ranges R	[0,20]	[0,20]			
$e_{\text{RMSE,super}}$	0.0815	0.0814			

Learning the weights: Using the most reliable future predictions improves the performance of the models. However, I select the future ranges manually depending on their reliability. Although it is possible to set a threshold for automatically selecting reliable future predictions, one still has to find the right threshold which gives the best performance.

A better solution is to learn the weights for the superposition. I use the model S1.2 from the previous experiments, which has shown good performance. With this model I predict the steering commands for the training and the test set. Then I use a small feed-forward neural network (shown in Fig. 4.5), which is trained on the predictions of the training set to predict the superimposed steering command, $\mathbf{c}_{\text{superNN},i}$. The result of applying this

Table 4.9.: Result of using a neural network to learn the weights for superposition applied on the model S1.2.

S1.2			
Superposition method	Error		
without superposition	$e_{\text{RMSE}}(r=0)$	=	0.0601
linear (all future predictions)	$e_{\text{RMSE,super}}$	=	0.0608
linear (most reliable future predictions)	$e_{\text{RMSE,super}}$	=	0.0562
non-linear (feed-forward neural network)	$e_{\text{RMSE,superNN}}$	=	0.0537

method is shown in Table 4.9. We see that the neural network learns to superimpose the future predictions and this method outperforms the previously presented linear methods.

The analysis of the video sequences in Fig. 4.10 with the steering predictions shows that this model is capable of smooth driving predictions, *i.e.* there is no heavy jitter. Furthermore, the prediction for the critical sharp turn to the left in the end of the sequence has improved and would not lead anymore to an accident, *c.f.* Fig. 4.6.



Figure 4.10.: Predictions of the model S1.2 for driving on the test track of the Udacity dataset. The black points on the circle visualize the predicted steering angles, whereas the white points shows the measured steering angles. In the left video we see the current prediction (smallest circle) and the future predictions (the bigger the circle, the more far the model looks ahead). The right video shows the superimposed steering predictions $\mathbf{c}_{\text{superNN,i}}$, of the same model generated by the feed-forward neural network.

4.3.4. Future predictions with different architectures

In this section I present the results of using three different architectures consisting of a CNN and a seq2seq model. I use the convolutional networks "ResNet-50", "Nvidia", and "Komanda" in combination with the seq2seq models Φ_{pro}, Φ_{simple}, and Φ_{E2D}, respectively.

Again we map a sequence of input frames $\mathbf{y}^{(t-\delta:t)}$ to a sequence of steering commands $\mathbf{c}^{(t:t+\tau)}$ as described in equation (4.7).

The experiments S2.1, S2.2 and S2.3 show a selection of models with different architectures. Experiment S2.1 works with a model consisting of the ResNet-50 CNN and the seq2seq model Φ_{pro} (Fig. 4.3b). The model of experiment S2.2 is trained with a Nvidia CNN and the simple seq2seq model Φ_{simple} (Fig. 4.3a). Finally, I present a combination of the spatiotemporal Komanda CNN and a seq2seq model with two-level encoder, Φ_{E2D}, which is shown in Fig. 4.3c. The hyperparamter settings can be found in Table 4.2.

Table 4.10 shows the prediction errors for the current and future time steps. Again, the test set is the Udacity test set `CH2_001`. All three experiments show the already seen monotonic increasing trend where the errors increase the more far the model predicts into the future (with an outlier at $r = 0$ for S2.1).

Experiment 2.1 shows that it is possible to finetune a very deep CNN like ResNet-50 on a driving task, although it is pre-trained on a different domain (ImageNet). We note, that training the ResNet needs strong augmentation of the data, otherwise it heavily overfits on the small driving dataset. Compared with the error of the single-frame ResNet-50 model in Sect. 4.3.1, we see that using the seq2seq module does not improve the prediction of the current frame. However, the future predictions for $r \leq 12$ are all statistically better than the prediction of the single-frame model. The analysis of the superposition of future angles with the current angle ($0 \leq r \leq 12$) shows that the error for the current frame can be reduced to $e^{(0)}_{\text{RMSE, super}} = 0.0749$.

The seq2seq module also improves the predictions in the experiment S2.2 where the convolutional layer structure of Nvidia is used. The error decreases from 0.01142 to 0.0892, a superposition of predictions for different ranges only provides marginal improvement: $e^{(0)}_{\text{RMSE, super}} = 0.0887$ for $0 \leq r \leq 6$.

The spatiotemporal CNN architecture of Komanda in combination with the seq2seq module, Φ_{E2D}, in experiment S2.3 shows comparable performance to the single-frame Nvidia model. Both models have a similar training error: $e_{\text{RMSE,train}} = 0.073$ for the seq2seq model with the Komanda CNN, and $e_{\text{RMSE,train}} = 0.081$ for the single-frame Nvidia model. I think that the Komanda model has too many parameters and overfits on the small dataset. More augmentation or more data could help to improve the predictions of this model. For this model superposition of the future predictions does not provide any improvement for the prediction of the steering angle at the current frame.

Table 4.10.: **Different architectures:** The second row lists which CNN extracts the features, whereas the third row in the table states the used seq2seq architecture (Φ_{pro}, Φ_{simple}, or Φ_{E2D}). The prediction range r (in frames) shows the temporal distance of the future prediction to the current frame.

S2.1 ResNet-50 Φ_{pro}		**S2.2** Nvidia Φ_{simple}		**S2.3** Komanda Φ_{E2D}	
r	e_{RMSE}	r	e_{RMSE}		
0	0.0958	0	0.0892	0	0.1189
4	0.0743	2	0.0901	2	0.1210
8	0.0795	4	0.0906	4	0.1227
12	0.0856	6	0.0922	6	0.1253
16	0.0926	8	0.0945	8	0.1285
20	0.1006			10	0.1321
24	0.1100			12	0.1359
28	0.1204			14	0.1400
32	0.1313			16	0.1440
36	0.1427			18	0.1484
40	0.1541			20	0.1530

4.4. Conclusion

This chapter shows first, that driving models without learning temporal structure, *i.e.* without recurrent layers, especially make poor predictions when information of previous frames is necessary to predict the correct steering angle. The seq2seq model does not have this problem (Fig. 4.10), because it is conditioned on a video sequence instead of a single frame.

Second, the section with future predictions of driving commands shows that predictions become less reliable the farther they lie in the future. This is expected for long-term predictions, because the model cannot predict the future steering angles for track sections it cannot see (*e.g.* because of a sharp turn). I show that a superposition of the most reliable future predictions reduces the error on the test set of Udacity. Simply superimposing all future predictions is not useful and leads to worse predictions. Since selecting the most reliable future predictions could be a sort of cherry picking, I propose to learn the weights for the superposition. A neural network can learn to predict a superimposed steering angle conditioned on the future predictions. This confirms that the superposition of future predictions from previous time steps improves the current prediction.

Finally, compared with the errors in the leader board of the Udacity challenge (Sect. 4.3.1), the results in Table 4.9 show that my superposition method outperforms the models of team Rambo and Chauffeur.

5. Visualizing & Understanding Driving Models

When artificial neural networks are used to control safety-critic systems like an autonomous car or driving assistants, customers and developers want to know if the system is reliable and what happens in a failure case. For that it is not possible to treat a neural networks as black box.

This chapter presents different approaches to visualize what a neural network has learned in the training process.

First, I visualize the embedded convolutional features of the Chauffeur CNN (from Sect. 3.4.2) to evaluate if the CNN has learned useful features for predicting steering angles. For that I use the t-distributed stochastic neighbor embedding (t-SNE) [66] to reduce the dimensionality of the CNN output feature vector to dimensionality 3 (Sect. 5.1.1). A clustering of frames (with the same steering angle as ground truth) in the three dimensional space would show that the CNN has learned features to predict steering commands.

Second, I want to explore the attention of the driving model to particular pixels in the input frame. To visualize which pixels lead to a high activation at the output, I use class activation maps (CAM) [95, 132] to highlight salient regions in the input (Sect. 5.1.2). Using this method, the evaluation of the driving model highly depends on the input data I use, *e.g.* if there are not red traffic lights along the evaluated road, it is not clear if the model has learned to recognize and understand the meaning of a red traffic light.

Therefore, I experiment with a method based on activation maximization (AM) [77, 29, 30], where I generate the preferred input for a driving model by maximizing the activation of one of its units (Sect. 5.1.3). With this method it is possible to visualize what the model has learned without using input data. However, without using a strong image prior most of the generated frames are not easily understandable for a human. Therefore, for video generation I present a spatiotemporal GAN which models the prior to generate natural video sequences (Sect. 5.1.4). In combination with an extension of the AM framework from [75] the videos, generated by maximizing different driving commands, are diverse, with high resolution, plausible, and easily understandable for

spectators. Additionally, I also propose methods to enable the model to generate videos of arbitrary length.

After describing the methods in Sect. 5.1, I explain in Sect. 5.2 the experimental setups (including implementation details) for each experiment. In Sect. 5.3 I show first the results for the well known approaches (t-SNE, CAM, and the AM without learned prior) before I present the novel experimental results which use the spatiotemporal GAN for video generation.

5.1. Methods

5.1.1. Visualization of embedded convolutional features

High dimensional feature vectors can be visualized using dimensionality reduction. Algorithms are for example principal component analysis (PCA) [121] or t-distributed stochastic neighbor embedding (t-SNE) [66, 111].

I apply t-SNE to visualize the output features of the CNN to evaluate if the learned features already present the driving task of predicting steering angles.

Before running the t-SNE algorithm I feed the CNN with the input frames of the test set (it is also possible to visualize the features of the training set). The predicted output is stored together with the ground truth. The feature vectors are used as input for the t-SNE algorithm, and the ground truth (*i.e.* the true steering angles) is later used to label the data points in the t-SNE visualization.

The t-SNE algorithm reduces the high dimensionality of the feature vector to a low-dimensional space in a two step process. First, it calculates the probability of data pairs (*i.e.* two feature vectors) being similar in the high-dimensional space. That means that for data point i a probability $p_{ij} \in P$ is computed which indicates how similar it is to the neighbor data point j. Second, the algorithm computes a probability distribution Q in the low-dimensional space minimizing the Kullback-Leibler divergence between data pairs using gradient descent.

The cost function C [66], which is minimized, is defined as

$$C = \sum_i KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ji}}{q_{ji}}, \quad p_{ij} \in P, q_{ij} \in Q \tag{5.1}$$

with $p_{ii} = q_{ii} = 0$ and where $KL(P||Q)$ denotes the Kullback-Leibler divergence between the joint probability distribution P of the high-dimensional space and the joint probability distribution Q of the low-dimensional space. For t-SNE the relations in the probability distributions are symmetric, *i.e.* $p_{ij} = p_{ji}$ and $q_{ij} = q_{ji}$ [66].

After performing the t-SNE algorithm the data points (*i.e.* the embedded convolutional features) are colorized and labeled depending on their steering angle (taken from the ground truth). This enables the viewer to evaluate if t-SNE created clusters of similar steering angles.

5.1.2. Class activation maps

One method for visualizing what pixels in the input image/video trigger a certain prediction of the neural network is to highlight the attention on pixels. This can be done with saliency maps [96] or class activation maps [132, 95].

For a saliency map the gradient of the output layer is computed w.r.t. to the input image. The result shows which pixels lead to a high activation in the output layer, and therefore are significant for the prediction. For saliency maps a CNN with fully-connected layers on top is a problem, because the spatial information is lost in the backpropagation of the gradient through the fully-connected layers. This problem is solved with the class activation map (CAM) algorithm, where the fully-connected layers are replaced by a global average pooling [132].

In case of a softmax output, the CAM M_c [132] for a class c is defined as:

$$M_c(x, y) = \sum_k w_k^{(c)} f_k(x, y), \tag{5.2}$$

with (x, y) being the spatial indices, $w_k^{(c)}$ being the weight between a unit k and the output class c, and the function $f_k(.)$ denoting the activation of the unit k.

However, the replacement with the global average pooling is a drawback, because retraining the model is necessary. The Gradient-weighted Class Activation Mapping (Grad-CAM) solves this problem by backpropagation of the gradient from the final convolutional layer conditioned on the output prediction [95].

To compute the Grad-CAM $\hat{M}_c$ [95], the weight $w_k^{(c)}$ in (5.2) is substituted with the gradient

$$\alpha_k^{(c)} = \underbrace{\frac{1}{Z} \sum_x \sum_y}_{\text{global average pooling}} \frac{\partial y^{(c)}}{\partial f_k(x, y)}, \tag{5.3}$$

with $y^{(c)}$ being the output of the final layer before the softmax.

The final equation for the Grad-CAM $\hat{M}_c$ [95] is

$$\hat{M}_c = \sum_k \alpha_k^{(c)} f_k(x, y). \tag{5.4}$$

By using for example a ReLU activation on $\hat{M}_c$, the Grad-CAM only highlight features which support the prediction of the class.

It is possible to compute the activation maps for a single class, but also for multiple classes. Visualizing multiple classes simultaneously is useful when we want to look at a group of targets: *e.g.* all animals in a ImageNet classification task or all turns (left and right) for a driving task. Furthermore, using a custom loss function the algorithm can also be modified for regression tasks.

5.1.3. Activation maximization without a learned prior

Class activation maps show what image features support a certain prediction and where the models focus at. However, one always need input data to compute the map. This holds the problem that the quality of the model analysis depends partially on the quality of the dataset we use for the evaluation. The idea of activation maximization is to generate preferred input data for the model by maximizing the activation on a chosen neuron [29]. This works by initializing the input data randomly and updating it iteratively with the backpropagated gradient of the chosen unit. In this way an image evolves out of the randomly-sampled data when using a CNN. The generated image shows high activation for the maximized unit. It is possible to generate images for the output units but also for units in hidden layers.

The optimization problem of the AM [127] with a regularization term $R_\Theta(\mathbf{x})$ is defined as

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmax}}(a_i(\mathbf{x}) - R(\mathbf{x})) \tag{5.5}$$

with $\mathbf{x}$ being the generated RGB input image, a_i being the activation for the maximized unit i with $\mathbf{x}$ as input, and $R(\mathbf{x})$ being the regularization function conditioned on $\mathbf{x}$.

The model is trained on natural images; therefore, regularization techniques are used to support a natural image generation with the AM [67, 127, 96]. I use the total variation (TV) loss $\mathcal{L}_{TV}$ [67] and a quadratic regularization loss $\mathcal{L}_{lp}$ [96], which helps bounding the pixel intensity. The overall loss $\mathcal{L}_{AM}$ of the AM algorithm is defined as

$$\mathcal{L}_{AM} = \lambda_c \mathcal{L}_c + \lambda_{TV} \mathcal{L}_{TV} + \lambda_{lp} \mathcal{L}_{lp} \tag{5.6}$$

with $\lambda_{(.)}$ being the weighting factors of the losses, and $\mathcal{L}_c$ being the loss at the condition model, the driving model.

The TV loss $\mathcal{L}_{TV}$ encourages the generation of natural images by minimizing the difference between neighboring pixels. Applied on an image the total variation regularizer $R_{TV}(\mathbf{x})$ [67] is defined as :

$$\mathcal{L}_{TV}(\mathbf{x}) = \sum_{i,j} \left((x_{i,j+1} - x_{i,j})^2 + (x_{i+1,j} - x_{i,j})^2 \right)^{\frac{\beta}{2}} \tag{5.7}$$

with the regularizer parameter β. Setting the parameter $\beta = 1$, can cause "spikes" in the generated image [14].

The quadratic regularization loss $\mathcal{L}_{lp}$ [96] is desribed as:

$$\mathcal{L}_{lp}(\mathbf{x}) = ||\mathbf{x}||_2^2 \tag{5.8}$$

5.1.4. Activation maximization using a spatiotemporal GAN

Spatiotemporal GAN

Our ST-GAN is a fully convolutional network with a feature vector $\mathbf{x}$ as input and a video sequence $G(\mathbf{x})$ as output. Instead of optimizing directly in the image space, Dosovitskiy *et al*. [25] designed a GAN trained with a composite loss to generate natural images optimizing in the latent space on the feature vector $\mathbf{x}$. We extend this GAN to the spatiotemporal domain for video generation. In addition to the adversarial loss $\mathcal{L}_{adv}$, a video loss $\mathcal{L}_{vid}$ and a feature loss $\mathcal{L}_{feat}$ are used to ensure the generation of natural videos. The optimization with the composite loss is defined as

$$\begin{aligned} G^*_{ST} = \arg\min_{G_{ST}} \max_{D_{ST}} & \lambda_{adv}\mathcal{L}_{adv}(G_{ST}, D_{ST}) \\ & + \lambda_{vid}\mathcal{L}_{vid}(G_{ST}) + \lambda_{feat}\mathcal{L}_{feat}(G_{ST}), \end{aligned} \tag{5.9}$$

with the spatiotemporal generator G_{ST} and the spatiotemporal discriminator D_{ST}. The discriminator is trained to differentiate real videos from fake (*i.e*. generated) ones. On the other side the generator trains on generating videos which can fool the discriminator. The overview of the training principle is shown in Fig. 5.1.

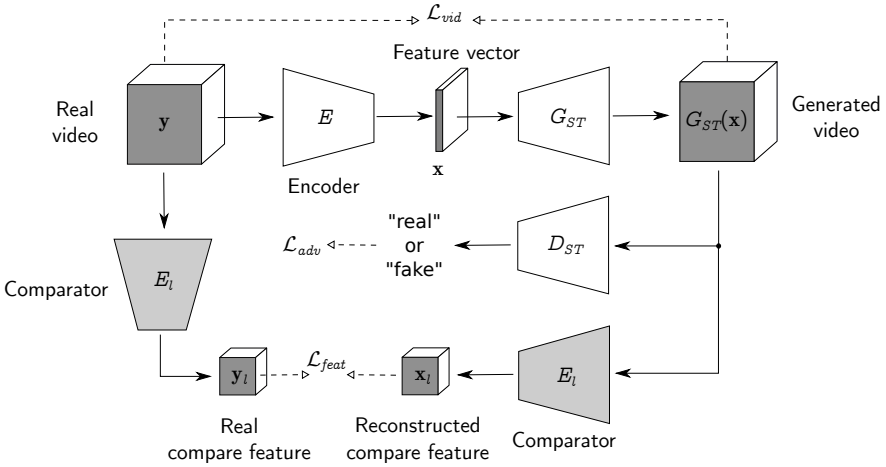


Figure 5.1.: Training principle for the spatiotemporal GAN, where the the generator G_{ST} has a code vector $\mathbf{x}$ as input and discriminator D_{ST} has a video $G(\mathbf{x})$ as input. The composite loss consists of three losses: $\mathcal{L}_{vid}$, $\mathcal{L}_{feat}$, $\mathcal{L}_{adv}$), which are used to train the generator G_{ST}. The encoder network E and the comparator network E_l are both AlexNets [56] pre-trained on ImageNet. The comparator is truncated at the last convolutional layer l.

The ST-GAN is trained on real videos $\mathbf{y}$, which are processed by an encoder E that outputs the feature vector $\mathbf{x} = E(\mathbf{y})$, the latent space for the optimization. The overall loss for the generator $\mathcal{L}_G$ is defined as:

$$\mathcal{L}_G = \lambda_{adv}\mathcal{L}_{vid} + \lambda_{feat}\mathcal{L}_{feat} + \lambda_{adv}\mathcal{L}_{adv}. \tag{5.10}$$

with $\lambda_{(.)}$ being weight factors for the individual losses.

The adversarial loss $\mathcal{L}_{adv}$ [38] is

$$\mathcal{L}_{adv} = -\log\Big(D_{ST}(G_{ST}(\mathbf{x}))\Big). \tag{5.11}$$

The video loss $\mathcal{L}_{vid}$ is a penalty on the color values of the video sequences:

$$\mathcal{L}_{vid} = \sum_{xyt} ||G(\mathbf{x}) - \mathbf{y}||_2^2, \tag{5.12}$$

with x, y describing the spatial dimensions and t being the temporal dimension of the generated video. The real video is denoted with $\mathbf{y}$.
The third loss is the feature loss $\mathcal{L}_{feat}$, which is an L_2 regression loss in the feature space of the comparator network E_l:

$$\mathcal{L}_{feat} = \sum_{xyt} ||E_l(G_{ST}(\mathbf{x})) - E_l(\mathbf{y})||_2^2, \tag{5.13}$$

where the comparator network E_l is truncated at the last convolutional layer l. For the comparator a image recognition network is used to provide robustness against deformations, viewpoints, or changing colors.

The feature loss $\mathcal{L}_{feat}$ is used, because the video loss $\mathcal{L}_{vid}$ can be ambiguous for variations in the RGB space (*e.g.* different colors or poses) and therefore, a high video loss would in this case not help to learn generating a better video.

While the the generator and the discriminator are spatiotemporal, *i.e.* convolutional in space and time, the encoder E and the comparator E_l work in the spatial domain, *i.e.* with two-dimensional filters. The encoder and decoder ensure the extraction and learning of features for natural video frame generation and are not responsible for learning temporal coherences.

The discriminator has to learn how to keep apart generated and real video sequences. Therefore, it has to analyze the image features in the video frames and also the temporal coherences between video frames. The standard adversarial loss $\mathcal{L}_{\mathcal{D}}$ [38] is used to train the discriminator:

$$\mathcal{L}_{\mathcal{D}} = -\Big[\log\Big(D_{ST}(\mathbf{y})\Big) + \log\Big(1 - D_{ST}(G_{ST}(\mathbf{x}))\Big)\Big]. \tag{5.14}$$

Architecture & Training

The generator G_{ST} and the discriminator D_{ST} are fully convolutional networks. Fig. 5.2 shows the architecture of the models, listing the number of channels, the padding, the kernel size and the stride. Additionally, in the last column it shows the output size of the layers. The stride is chosen in a way to keep the sequences of the same temporal length throughout the network. Our architecture is based on the GAN of Dosovitskiy *et al*. [25], which extend for the spatiotemporal domain. To prevent overtraining of G_{ST} on D_{ST}, the discriminator has the the generated/real video and the encoded feature vector x as input.

First, we fine-tune the image generator of [25] as well as the discriminator on the target task of the condition network, *i.e*. the driving task. We use an ImageNet pretrained AlexNet [56] as encoder E truncated at the layer fc6. It is also possible to invert from other layers, but [25] shows that the fc6 layer provides the best results for image generation. The same goes for the comparator network E_l where we truncate the AlexNet at layer pool5 [25].

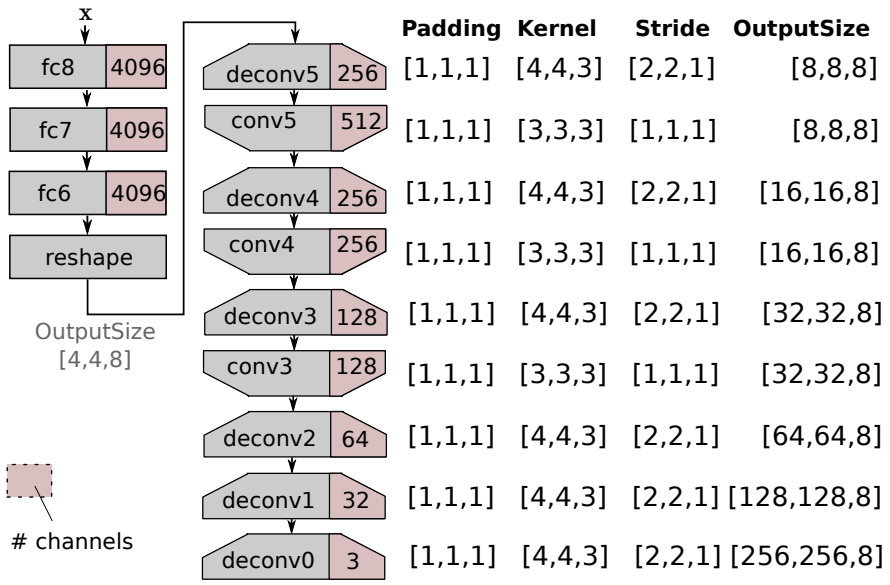
Second, we transfer the parameters of the fine-tuned image generator/discriminator to our spatiotemporal models. The parameters of the fully-connected layers and the biases of the convolutional layers can be directly copied from image to video space (except of the last fully-connected layer of the discriminator which is randomly initialized), whereas the weights of the convolutional layers are expanded to a three-dimensional matrix following [31], where the filters are initialized as kernels that average in time:

$$\hat{\mathbf{w}}_l(x, y, t, c) = \frac{\mathbf{w}_l(x, y, c)}{T'}, \forall t \in [1, T'], \tag{5.15}$$

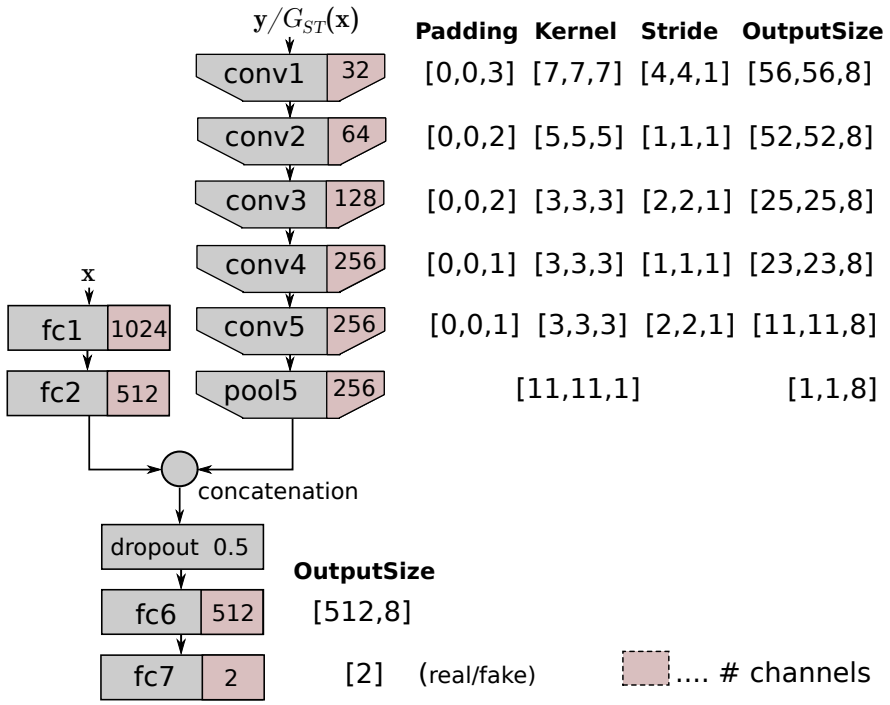
where (x, y, t, c) describe the spatiotemporal filter of the video cube in the RGB space, whereas (x, y, c) donate the spatial filter dimensions for the RGB image space. The parameter T' donates the number spatiotemporal filters in the layer, *i.e*. the length of the video sequence that is generated by applying the filter.

The training of the ST-GAN starts with this transfer-learned model from the two-dimensional image space. Video sequences of length T are encoded by E and fed into the generator G_{ST}. The generator learns spatiotemporal features for producing plausible videos with motion and natural image features.

Nevertheless, the length of the video generated by the ST-GAN is fixed. This is also a limitation of other fully convolutional generators as used in previously designed video GANs [115, 116, 92]. Therefore, we propose two approaches for long-term video generation where sequences of arbitrary length can be produced.



(a) Spatiotemporal generator G_{ST}



(b) Spatiotemporal discriminator D_{ST}

Figure 5.2.: ST-GAN architecture. Spatiotemporal dimensions are shown in $[x, y, t]$; training duration is set to $T = 8$.

Generation of Long Sequences

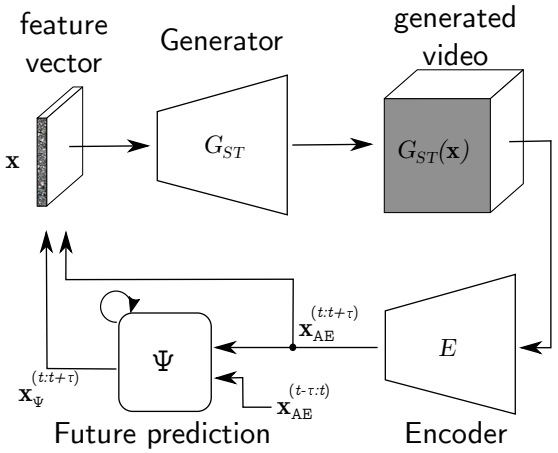


Figure 5.3.: Long-term video generation with the autoencoder structure. The model consists of the spatiotemporal generator G_{ST}, the encoder E and the future prediction module Ψ that is conditioned on the past.

The ST-G can generate short-term sequences of a certain length, which is determined by the available memory size of the used GPU. For longer sequences it is necessary to condition the new sequence on the last frames of the previous sequence. Otherwise the generator will create a natural looking video sequence, which will not fit to the previous generated sequence – the transition is not smooth.

For the generation of long-term videos, we introduce a differential auto-encoder (AE) with the future prediction module Ψ to condition video samples on a previously generated sequence. With this method, we achieve that the ST-GAN generates smooth video transitions where consecutive sequences fit to each other. The approach is shown in Fig. 5.3. The video generation is conditioned on the previously generated auto-encoded sequence $\mathbf{x}_{\text{AE}}^{(t-tau:t)}$ and optionally the current auto-encoded sequence $\mathbf{x}_{\text{AE}}^{(t:t+\tau)}$. The new sequence with T frames is generated by updating the previous code vector from our ST-GAN by

$$\begin{aligned}\Delta\mathbf{x}^{(t:t+T)} = \epsilon_{\text{AE}}(\mathbf{x}_{\text{AE}}^{(t:t+T)} - \mathbf{x}^{(t:t+T)}) + \mathcal{N}(0, \epsilon_{\text{noise}}^2) \\ + \epsilon_{\Psi}(\mathbf{x}_{\Psi}^{(t:t+T)} - \mathbf{x}^{(t:t+T)}) \odot \sigma(T)\end{aligned} \tag{5.16}$$

with $\epsilon_{(.)}$ being the weights, $\mathbf{x}_{AE} = E(G_{ST}(\mathbf{x}))$ the encoded generated video, $\mathbf{x}_{\Psi}^{(t:t+T)}$ is the code vector generated by the future prediction module. The operation $\odot$ is an element wise multiplication and $\sigma(T)$ exponentially decays the temporally distant feature vectors in the sequences generated by the future prediction module. The zero-mean Gaussian noise $\mathcal{N}(0, \epsilon_{\text{noise}}^2)$ has a unit standard deviation. If $\epsilon_{\text{noise}}^2 > 0$, the equation (5.16) changes the model to a denoising AE and diverse videos that are conditioned on the past sequence are generated.

We propose two different approaches of AEs, which condition the video generation on the past: temporal convolution Ψ_{tconv}, a seq2seq approach where we differentiate between two versions (conditioned on the future and past Ψ_{seq2seq}, and conditioned only on the past Ψ'_{seq2seq}).

Temporally convolutional AE: As baseline we use a **temporal convolution** approach Ψ_{tconv}. The feature vector entries of the last frames $\mathbf{x}^{t-\tau:t}$ are used as initialization of the first feature vector entries ($\mathbf{x}^{(t:t+\tau)}$) of the new video sequence; the remaining entries are initialized like the first sequence $\mathbf{x}^{(0:T)}$ (*e.g.*, randomly):

$$\mathbf{x}^{(t:t+\tau)}_{\Psi,\text{tconv}} = \mathbf{x}^{t-\tau:t}, \tag{5.17}$$

where τ is the temporal overlap between two sequences as shown in Fig. 5.4. Since we temporally convolve the ST-GAN over the generated sequence codes, with temporal overlap $\tau < T$, this approach inherits the spatiotemporal appearance from the past sequence and implicitly creates transitions by ST-GAN's 3D filters.

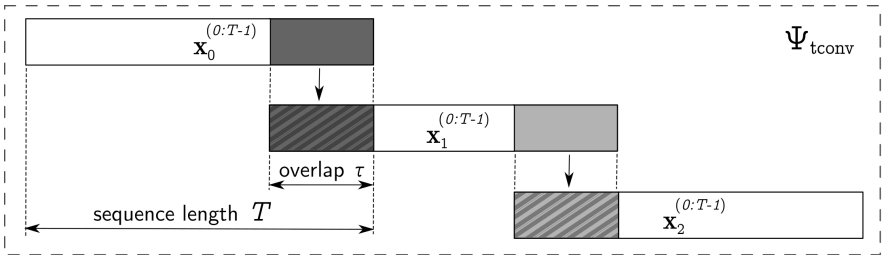


Figure 5.4.: Temporal convolution approach where the last code vectors of the previous sequences are used for the initialization of the first code vectors of the future video sequence.

Sequence-to-sequence AE: The temporal convolutional approach has the drawback that it has low performance regarding computation time due to the overlap of consecutive sequences. A more elaborate approach is the seq2seq AE which predicts future sequences conditioned on the past. We implement two similar variants of the seq2seq approach, where the first one Ψ_{seq2seq} is a seq2seq RNN that takes the code vector of the past sequence and transforms the future code such that it maps the past. The second variant Ψ'_{seq2seq} only takes the code vector of the past sequence to predict the future code.

Future-past conditioned module Ψ_{seq2seq}:
The future-past conditioned approach Ψ_{seq2seq} has the advantage that it does not only condition the future prediction on the past sequence $\mathbf{x}_{AE}^{(t-\tau:t)}$ but also transforms the future code $\mathbf{x}_{AE}^{(t:t+\tau)}$ in the AM algorithm. Fig. 5.3 shows an illustration of the approach. The conditioning on the past creates a prediction with a smooth transition between the previous samples and the future sequences. The prediction of the seq2seq module at time t is

$$\mathbf{x}_{\Psi,\text{seq2seq}}^{(t:t+\tau)} = \text{LSTM}\Big(\mathbf{x}_{\text{AE}}^{(t-\tau:t)}, \mathbf{x}_{\text{AE}}^{(t:t+\tau)}; \mathbf{\Psi}_{\text{LSTM}}\Big), \tag{5.18}$$

with τ being the temporal sequence length which is processed by the LSTM which has the parameters $\mathbf{\Psi}_{\text{LSTM}}$. The encoded past is denoted as $\mathbf{x}_{AE}^{(t-\tau:t)} = E(G_{ST}(\mathbf{x}^{(t-\tau:t)})$, whereas the encoded future is $\mathbf{x}_{AE}^{(t:t+\tau)} = E(G_{ST}(\mathbf{x}^{(t:t+\tau)})$.

Past conditioned module Ψ'_{seq2seq}:
This variant uses an RNN which is taking the code of the past sequence, $\mathbf{x}_{AE}^{(t-\tau:t)}$, as input and predicts the future code $\mathbf{x}_{\Psi',\text{seq2seq}}^{(t:t+\tau)}$:

$$\mathbf{x}_{\Psi',\text{seq2seq}}^{(t:t+\tau)} = \text{LSTM}\Big(\mathbf{x}_{\text{AE}}^{(t-\tau:t)}; \mathbf{\Psi}'_{\text{LSTM}}\Big), \tag{5.19}$$

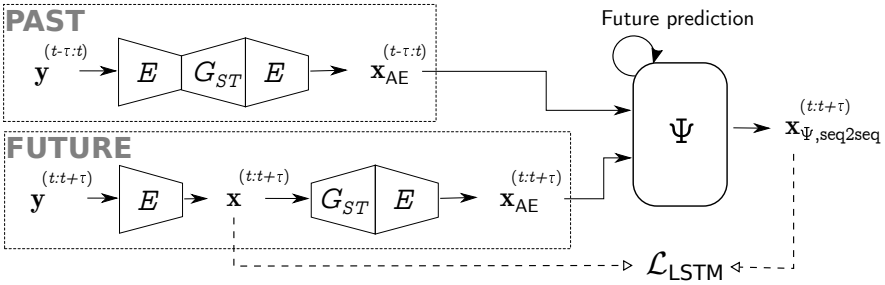
with the model parameter $\mathbf{\Psi}'_{\text{LSTM}}$. The encoded past is denoted as $\mathbf{x}_{AE}^{(t-\tau:t)} = E(G_{ST}(\mathbf{x}^{(t-\tau:t)})$.

Training the future prediction module:
The future prediction module consist of an LSTM which is trained on the the same video sequences as used for training the ST-GAN. For Ψ_{LSTM} fhe real videos $\mathbf{y}^{(t-\tau:t+\tau)}$ for past and future sequence are auto-encoded using the ST-GAN and the encoder E (the ImageNet pretrained AlexNet [56]) and fed to the LSTM as shown in Fig. 5.5a. For Ψ'_{LSTM} only the auto-encoded feature vector of the past sequence $\mathbf{y}^{(t-\tau:t)}$ is used as input for the LSTM (see Fig. 5.5b).

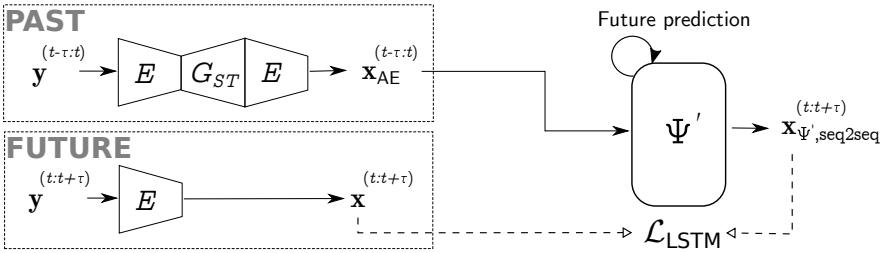
The seq2seq modules are optimized using an L2 loss in the feature space of the auto-encoder:

$$\mathcal{L}_{\text{LSTM}} = \sum_t ||\mathbf{x}_{\Psi,\text{seq2seq}}^{(t:t+\tau)} - E(\mathbf{y}^{(t:t+\tau)})||_2^2, \tag{5.20}$$

with t indexing the temporal feature dimensions of the predicted future code vector $\mathbf{x}_{\Psi,\text{seq2seq}}^{(t:t+\tau)}$ and the real feature vector $E(\mathbf{y}^{(t:t+\tau)})$.



(a) Future-past conditioned module Ψ_{seq2seq} with the autoencoded past $\mathbf{x}_{\text{AE}}^{(t-\tau:t)}$ and the autoencoded future $\mathbf{x}_{\text{AE}}^{(t:t+\tau)}$ as input and the real future $\mathbf{x}^{(t:t+\tau)}$ as target.

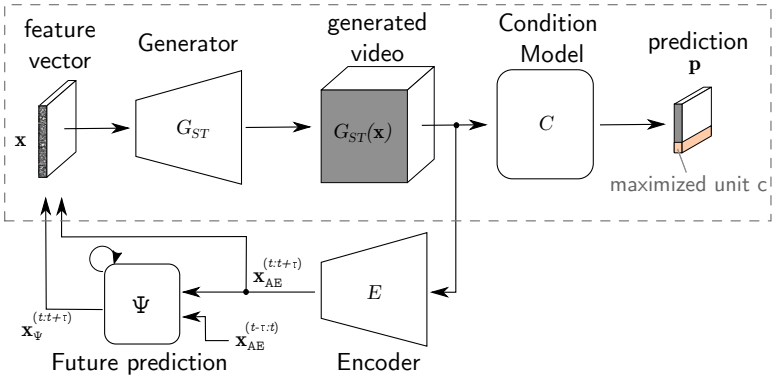


(b) Past conditioned module Ψ'_{seq2seq} which takes the autoencoded past $\mathbf{x}_{\text{AE}}^{(t-\tau:t)}$ as input and the real future $\mathbf{x}^{(t:t+\tau)}$ as target.

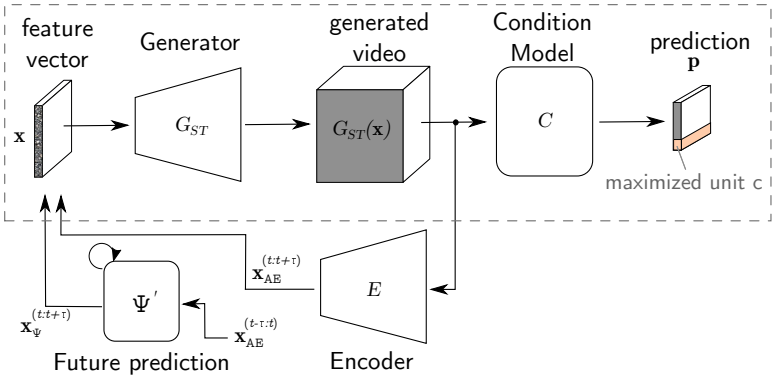
Figure 5.5.: Training framework of the seq2seq modules, where the future prediction module is optimized with the loss $\mathcal{L}_{\text{LSTM}}$ from equation (5.20).

Spatiotemporal activation maximization

For visualizing and understanding neural networks, we apply our ST-GAN within the activation maximization (AM) framework for generating preferred videos of multiple condition models. By backpropagation and gradient accent on the code vector $\mathbf{x}$ the AM produces sequences which maximally activate the neuron c of the condition model C. Our method is an extension of the Plug&Play Generative Network framework of [75] which builds on [74]. Instead of generating images, our approach uses the AM framework in the spatiotemporal domain and generates videos as shown in Fig. 5.6. It shows both AM frameworks with the two different future prediction modules Ψ_{seq2seq} and Ψ'_{seq2seq}.



(a) AM with past-future conditioned module Ψ_{seq2seq}



(b) AM with past conditioned module Ψ'_{seq2seq}

Figure 5.6.: Spatiotemporal activation maximization with the spatiotemporal generator G_{ST} and the future prediction module Ψ in the AE path. The future prediction module enables the ST-GAN to produce long-term videos with smooth transitions between the generated sequences. The video generation maximizes the activation for unit c in the condition model C.

Short-term AM: The input of the AM framework is the code vector $\mathbf{x}$ of the temporal length of T frames. It is initialized randomly: $\mathbf{x}_0^{(0:T-1)} = \mathcal{N}(0,1)$ with a zero-mean Gaussian noise with unit standard deviation. After predicting the softmax prediction $\mathbf{p}$ of the condition model C from the generated video of the ST-GAN, the selected unit c which should be maximized is multiplied with a unity vector. For this multiplied vector at the output, the gradient is backpropagated to the code vector $\mathbf{x}$ which is updated via gradient ascent:

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmax}} \langle C(G_{ST}(\mathbf{x})), e_c \rangle, \tag{5.21}$$

with e_c being the natural basis vector of the same shape as the prediction vector. The vector has an entry with value 1 at the position of the maximized unit c. As the code vector only changes slightly with each AM iteration, the update is applied until the chosen unit c shows maximal activation at the truncated model C_c by backpropagating the gradient $\frac{\partial \log C_c(G_{ST}(\mathbf{x}))}{\partial \mathbf{x}}$ until it converges or a maximal number of iterations is reached.

Long-term AM: Beside short-term sequences we also want to produce videos of arbitrary length. Using the temporal convolution or the seq2seq approach, we can generate long-term video sequences. The future prediction modules are incorporated in the auto-encoder path of the ST-GAN as shown in Fig. 5.6. They are responsible that consecutive sequences fit together naturally. We can directly extend the update equation (5.16) of the temporal autoencoder and perform optimization to excite a chosen unit c of the condition model C:

$$\begin{aligned}\Delta\mathbf{x}^{(t:t+T)} &= \epsilon_{\text{AE}}(\mathbf{x}_{\text{AE}}^{(t:t+T)} - \mathbf{x}^{(t:t+T)}) + \mathcal{N}(0, \epsilon_{\text{noise}}^2) \\ &+ \epsilon_{\Psi}(\mathbf{x}_{\Psi}^{(t:t+T)} - \mathbf{x}^{(t:t+T)}) \odot \sigma(T) \\ &+ \epsilon_c \frac{\partial \log C_c(G_{ST}(\mathbf{x}^{(t:t+T)}))}{\partial G_{ST}(\mathbf{x}^{(t:t+T)})} \frac{\partial G_{ST}(\mathbf{x}^{(t:t+T)})}{\partial \mathbf{x}^{(t:t+T)}},\end{aligned} \tag{5.22}$$

where ϵ_c weights the gradient for maximizing unit c in the condition model C. The rest is identical to equation (5.16). A larger ϵ_{AE} favors the generation of novel sequences with similar content, whereas the Gaussian noise controlled by ϵ_{noise} stimulates the generation of diverse images.

In the experiments we apply our approach for visualizing and understanding deep driving architectures. We use different driving models and also show experiments where we condition on multiple networks (driving model and scene model).

5.2. Experimental setup

This section describes the setups of the experiments and also gives implementation details, like hyperparameter settings.

5.2.1. CNN feature embedding with t-SNE

I visualize the output features of the Chauffeur CNN (architecture in 3.4.2) using the t-SNE algorithm. The output feature vector of CNN has the shape (1×8960), which is is reduced to 3D.

I use the build-in embedding projector of TensorBoard [1] to compute the t-SNE. Therefore, after extracting the feature vectors with TensorFlow [1], they are stored in the log file of TensorBoard. Additionally, I save a meta file containing the labels with the ground truth.

Parameters of the t-SNE analysis of the Chauffeur-CNN output features in TensorBoard:

- Dataset: Udacity test set `CH2_001` (described in Sect. 3.3.1)
- Number of data points = 5279
- Feature dimension = 8960
- Visualization dimension = 3
- Perplexity = 42
- Learning rate = 107
- Number of iterations = 106

5.2.2. Class activation maps for driving models

We use the framework Keras-Vis [55] for generating Grad-CAM heatmaps to visualize multiple driving models. Although Keras-Vis is built for models implemented in Keras [17], it is also possible to visualize models which are purely implemented with TensorFlow after modifying the framework slightly.

For classification tasks the framework already offers all necessary functions to compute the CAM. The models trained on the Udacity dataset have a linear output for the steering angle. Therefore, we use a custom loss $\mathcal{L}_{\text{reg}}$ to modify the framework for the regression task:

$$\mathcal{L}_{\text{reg}} = ||C(\mathbf{y}) - \mathbf{c}_{\text{meas}}||_2^2, \tag{5.23}$$

with $C(\mathbf{y})$ being the prediction of the driving model on the video sequence $\mathbf{y}$ and $\mathbf{c}_{\text{meas}}$ being the measured steering commands from the dataset.

Keras-Vis provides a parameter to set a gradient modifier. This modifier can compute absolute values of the gradient, add a ReLU activation, or negate the gradient values. For regression tasks this can be used to visualize the attention of pixels which increase or decrease the loss.

Driving models: We analyze the Nvidia and the ResNet-50 model which use a single frame as input and the Chauffeur-seq2seq model with five frames input (model S1.3 from Chapter 4). Further experiments show visualizations of the BDD driving models. Detailed description of the driving models can be found in Sect. 3.4.

The models trained on the Udacity dataset – Nvnida, ResNet-50, Chauffeur-seq2seq – are tested them on the test track of this dataset (with a sampling rate of 2fps). All these models have a linear output unit for the steering angle. Therefore, we use the custom loss of (5.23). For the base experiments we use the "absolute" gradient modifier, but we also investigate how unprocessed and negated gradients change the CAMs.

The BDD models are trained on the BDD-V dataset and we also test the models on video sequences from the validation set. We experiment with the discrete models (BDD-CNN and FCN-LSTM) and the continuous BDD model. The BDD models are not

implemented in Keras, however it is also possible to visualize pure TensorFlow models in Keras-Vis. All BDD models treat the driving task as classification problem, *i.e.* they calculate a softmax over the possible predictions. We use the gradient modifier 'ReLU' to show heatmaps where pixels which contribute to the prediction show high attention.

5.2.3. Activation maximization with Udacity driving models

Beside a framework to generate CAMs, Keras-Vis [55] also provides an activation maximization framework for Keras models. I use this framework to generate input images which maximize convolutional filters of the Chauffeur CNN from Sect. 3.4.2.

For the optimization I use the Adam optimizer [53] with a learning rate of 10^{-5}. The weights for the different losses are set to $\lambda_c = 0.01$, $\lambda_{lp} = 10$, and $\lambda_{TV} = 0.01$. The activation maximization algorithm runs for 500 iterations.

5.2.4. Generating inputs using the spatiotemporal GAN

We apply our ST-GAN for visualizing and understanding deep driving models. Additionally to the BDD models [124] from Sect. 3.4.3, we also visualize two models from Sect. 3.4.1 which are trained on the Udacity dataset. We qualitatively evaluate the driving models and look for strengths and weaknesses using the spatiotemporal AM framework with our ST-GAN.

Overview of experiments:

1. *Short-term (Sect. 5.3.4):* We verify if our ST-GAN generates realistic short-term videos for different driving commands for the discrete BDD models [124]. We also compare the results to video sequences produced by the image generator.
2. *Long-term (Sect. 5.3.5):* We evaluate our long-term approaches and generate long driving videos maximizing different driving commands.
3. *Continuous models (Sect. 5.3.6):* With continuous models the steering angle can be predicted more precisely. This experiment shows if the ST-GAN can also generate plausible videos for these kind of driving models.
4. *Hidden layers(Sect. 5.3.7):* Beside maximizing units at the output layer, we also visualize the units at the hidden layers of the driving model. This gives more insights, what particular layers have learned. Furthermore, the generated videos of the fully-connected layers are clustered in groups depending on the content of the sequences.
5. *Conditioning on multiple models (Sect. 5.3.8):* We generate driving sequences which are conditioned on the driving model and an additional scene model for changing the driving scenery.

6. *Initializing from an image Sect. 5.3.9:* In this experiment we explore how the initialization of the input with the code vector of an image influences the generation of the driving video.

Please note that in Appendix B we provide plots of video sequences unrolled in time. In the online version of this thesis the videos can be played on click in Adobe Reader.

Data: We use the BDD-V dataset [124], which is described in Sect. 3.3.4, for training the ST-GAN. We sample the dataset with 3 fps, because that is also the sampling frequency of the BDD driving models [124].

Driving models: As condition models C, we use driving models trained for predicting future vehicle egomotion. We generate videos using the AM framework as shown in Fig. 5.6. We experiment with the BDD driving models [124], the Nvidia model [6], and the ResNet-50 model [42]. More details to the models can be found in Sect. 3.4.3 and Sect. 3.4.1, respectively.

Implementation details

ST-GAN: For training the ST-GAN we use a data reader in TensorFlow [1] to load the driving sequences. We set the frame rate of the data reader to 3fps, which is the same sampling rate as for the analyzed BDD driving models [124]. The ST-GAN itself is implemented in Caffe [48]. We use the Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$ [53]. First, we fine-tune the image generator/discriminator from [25] on the driving task for 140,000 iterations with a mini-batch size of 64 and L2 regularization of $4 \cdot 10^{-4}$. The base learning rate is $5 \cdot 10^{-5}$, reduced by 50% after [10k, 25k, 50k, 70k, 90k] iterations.

Second, we transfer the weights from the spatial to the spatiotemporal domain, where we set the sequence length to $T = 8$ frames, *i.e.* we train on driving sequences of 2.66 seconds. The ST-GAN is trained for 160,000 iterations with a mini-batch size of 8 and an initial learning rate of 10^{-5} with L2 regularization of $4 \cdot 10^{-4}$. The learning rate decays by half after $[10k, 25k, 50k, 70k, 90k]$ iterations.

The weights for the composite loss of the generator in (5.10) are $\lambda_{adv} = 100$, $\lambda_{vid} = 2 \cdot 10^{-6}$, and $\lambda_{feat} = 1 \cdot 10^{-2}$.

For both training procedures we follow the heuristic rule for pausing the training of generator or discriminator depending on the loss ratio [25]: $r = \mathcal{L}_D \big/ \mathcal{L}_G$, where $\mathcal{L}_D$ is the loss of the discriminator in (5.14) and $\mathcal{L}_G$ is the loss of the generator in (5.10). If the ratio drops below 0.1, we pause the discriminator training. For a ratio $r > 10$ the training of the generator is stopped. For $0.1 < r < 10$ the generator and the discriminator are trained.

Future prediction LSTM: The seq2seq model Ψ_{seq2seq} for long-term video generation consists of an LSTM encoder and an LSTM decoder. Each LSTM layer has 128 output neurons and uses $\tanh$-activation. The seq2seq model is illustrated in Fig. 5.7. The seq2seq model is implemented in Keras [17] with TensorFlow [1].We use the RMSprop optimizer with $\rho = 0.9$, $\epsilon = 1e-8$, and a starting learning rate of $1e-6$. The gradient is clipped at the value 10. We train the LSTM for 140,000 iterations with a mini-batch size of 12 and a temporal depth $\tau = 8$ frames. The learning rate is reduced by 50% when the validation loss plateaus for longer than 7,000 epochs. We only save the best model measured on the validation loss. The code vectors at the input of the LSTM are normalized to zero-mean and unit standard deviation, with the statistics calculated over the entire training set.

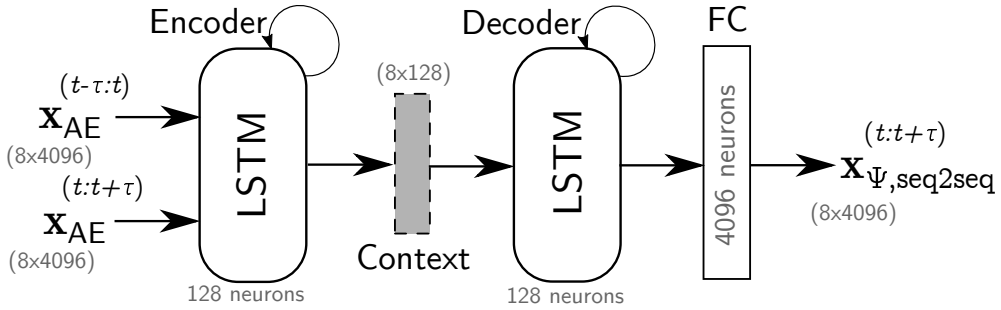


Figure 5.7.: Seq2seq model architecture with two LSTM layers and one fully connected (FC) layer. The temporal depth is $\tau = 8$ frames. The code vectors have a dimension of 4096.

Parameters of the AM experiments: Table 5.1 shows the AM parameters of all conducted experiments. We use a step size of 0.3 for all experiments with a linear decay to 0.01 for the updates in eq. (5.22).

Table 5.1.: This table shows the parameters of the experiments: the weightings parameters $\epsilon_{(.)}$ of the update equation (5.24), the sequence length T in frames, and the number of AM iterations N. The parameter τ denotes the overlap (Ψ_{tconv}) or the temporal depth (Ψ_{seq2seq}). Empty entries mean the parameter is zero.

Sect.	**Experiment**	**Parameter**							
		N	T	ϵ_{AE}	ϵ_c	ϵ_{noise}	ϵ_Ψ	$\epsilon_{c,\text{scene}}$	τ
5.3.4	four directions	200	8	10^{-5}	1	10^{-17}			
5.3.4	discrete models	100	32	10^{-5}	1	10^{-17}			
5.3.5	short-term	50	32		1	10^{-17}			
5.3.5	Ψ_{tconv}	50	16	10^{-5}	1	10^{-17}			8
5.3.5	Ψ_{seq2seq}	50	16	10^{-5}	1	10^{-17}	10^{-4}		8
5.3.5	Ψ'_{seq2seq}	50	16	10^{-5}	1	10^{-17}	10^{-4}		8
5.3.6	continuous models	50	32	10^{-5}	1	10^{-17}	10^{-4}		8
5.3.7	hidden layer	50	32	10^{-5}	1	10^{-17}			
5.3.8	scenery change	50	32	10^{-5}	1	10^{-17}	10^{-4}	$5 \cdot 10^{-3}$	
5.3.9	initialization with image	50	32	10^{-5}	1	10^{-17}	10^{-4}		8

5.3. Experimental results

5.3.1. CNN feature embedding with t-SNE

At the start of the t-SNE algorithm, a starting distribution is calculated and the data points of the feature vectors form a spherical point cloud. With each iteration of t-SNE the data points of feature vectors with big steering angles (left or right) move from the center to the surrounding. Data points with small steering angles (*i.e.* for driving straight) stay more in the center of the point cloud.

The result is shown in Figure 5.8, where the numbers/colors of the data points indicate different ranges of steering angles. Table 5.2 lists the labels with the corresponding steering angle ranges and steering directions.

This experiment shows that the learned output features of the CNN can be clustered in groups depending on the steering angle. We see that the CNN produces a meaningful output which is already conditioned on the driving task. That indicates that another neural network can learn to process the feature vectors and predict the steering commands.

Table 5.2.: Labels and colors of data points for the t-SNE visualization in Figure 5.8. Each label indicates a different range of steering angles. For example, the label 0 with color red is a group of feature vectors of images, where the car goes left.

Label	**Color**	**Steering angle**		**Direction**
		Lower bound	**Upper bound**	
0	red	0.50	∞	strong left
10	light-blue	0.25	0.50	left
20	gray	-0.25	0.25	straight
30	turquoise	-0.50	-0.25	right
40	green	$-\infty$	-0.50	strong right

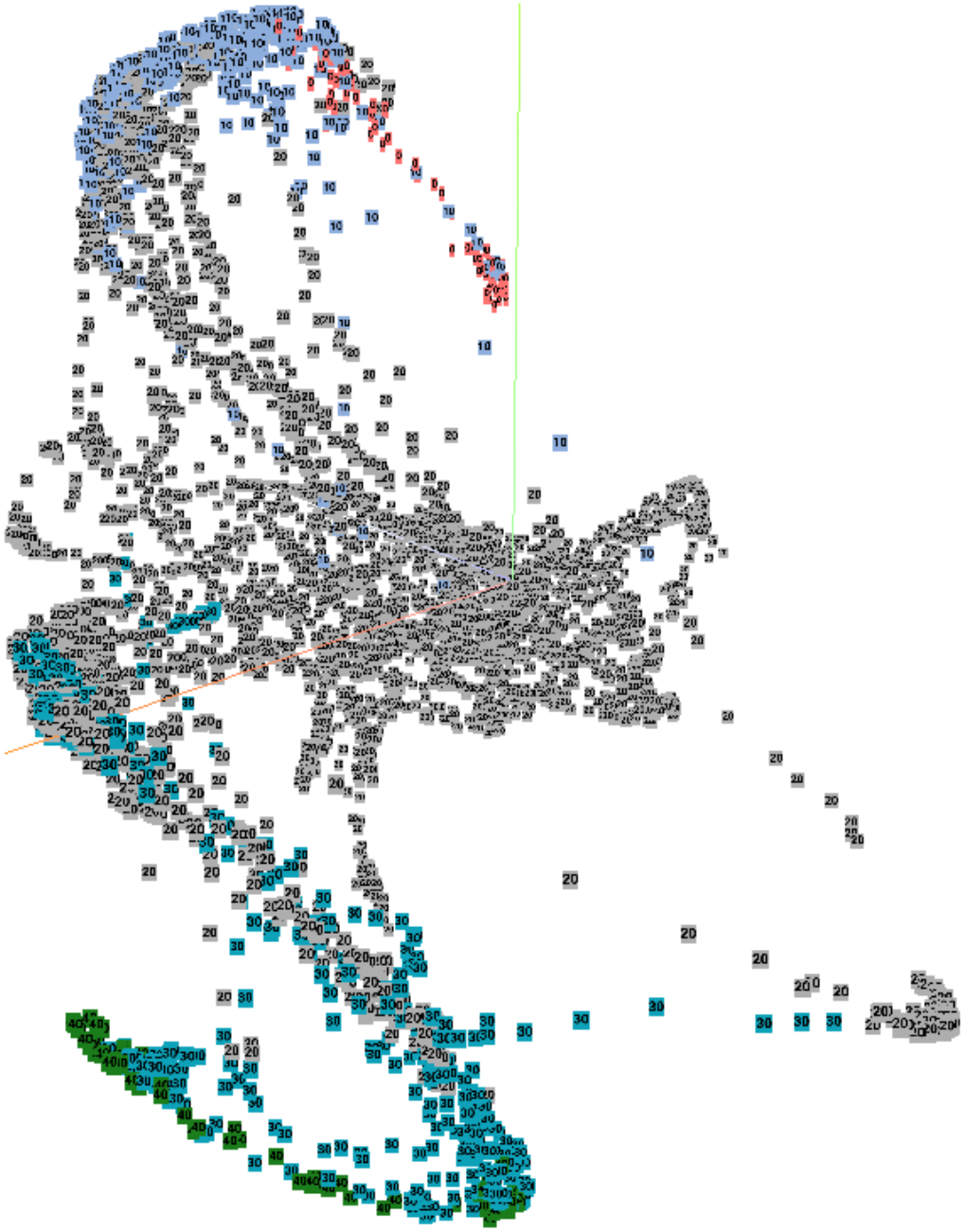


Figure 5.8.: Visualization of t-SNE dimensionality reduction to 3D. The labels/colors indicate the steering angle which corresponds to the feature vector of the image frame (see Table 5.2 for details). The red data points (label 0) belong to images of left turns. The big cluster in gray (label 20) contains features of images where the car goes straight. A turn to the right is showed in images with the data point colored in green (label 40).

5.3.2. Class activation maps for driving models

In this section we show the results of the analysis of multiple driving models using grad-CAM. The output of the grad-CAM algorithm is visualized in heatmaps, where the color represents the attention (see Fig. 5.9).



Figure 5.9.: Colormap for heatmaps to visualize the attention on pixels.

Driving models trained on the Udacity dataset

I start by visualizing driving models with single-frame input trained on the Udacity dataset. The road in the video is the first part of the test track of the Udacity Challenge 2 as shown in Fig. 5.10a.

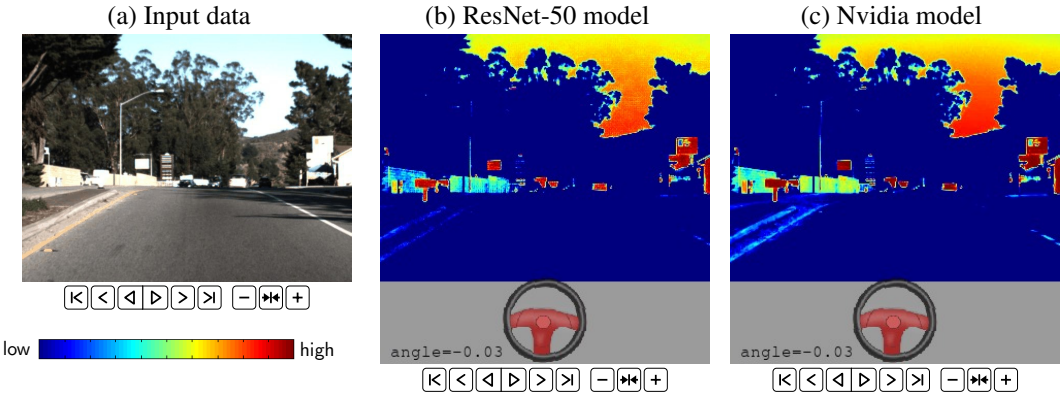


Figure 5.10.: CAM analysis for models with single-frame input. The displayed angle is the prediction of the model for the input data shown in (a).

Fig. 5.10b and 5.10c show the heatmaps using the Nvidia model [6] and the ResNet-50 [42]. Already in the first frame we notice that both models highly overfit on the skyline. This happens because we train these models on a small dataset, *i.e.* the model has learned to memorize the skyline for the steering command prediction during the training process. However, we see that the models have high attention values for road markings (lines, but also arrows on the asphalt). The models also recognize other vehicles and traffic signs. Furthermore, they base the prediction of the steering commands also on the appearance of reflector posts and power poles.

The next model uses multiple input frames which are processed by the Chauffeur CNN (see Sect. 3.4.2). With the sequence of extracted features a seq2seq module predicts the current and future steering commands. We note, that only the prediction for the current time step is used for generating the heatmap. This model is also trained on the Udacity dataset. To avoid the overfitting on the sky this model only uses the lower half of the input frames. Thus, the model only sees the street and some surroundings as shown in

Fig. 5.11. We recognize that this seq2seq model shows more attention on the road itself and not only the road markings (which still have a very high attention value). Also this model recognizes other cars and road elements like reflector posts. We notice that this model has problems with shadows on the road, which appear with very low attention values (in dark blue). This is a problem, because the shadows partially obstructs that the model pays attention on road markings at the side of the road, which are potentially important for the lane holding task in autonomous driving.

For all three models we recognize that at frame 62 the steering prediction is difficult, because there is a strong transition from shadow to bright light and back to shadow (see Fig. 5.12a). In this scene the Nvidia model has the problem that it makes the steering prediction based only on a single frame. This area of bright light between the shadows attracts the attention of the model, and a false prediction happens. The Chauffeur-seq2seq model overcomes this problem, because it can use the information of multiple input frames. As shown in Fig. 5.12 at this time step the Nvidia model pays attention to the light structure, which does not give information for the driving task, whereas the Chauffeur-seq2seq model makes use of its other input frames which contain useful features (*e.g.* road markings). In the videos we observe that the Nvidia model does not make a big mistake in the prediction. However, using only a single frame for driving predictions is potentially dangerous, because the model cannot learn temporal coherences or smooth driving without sudden strong direction changes.

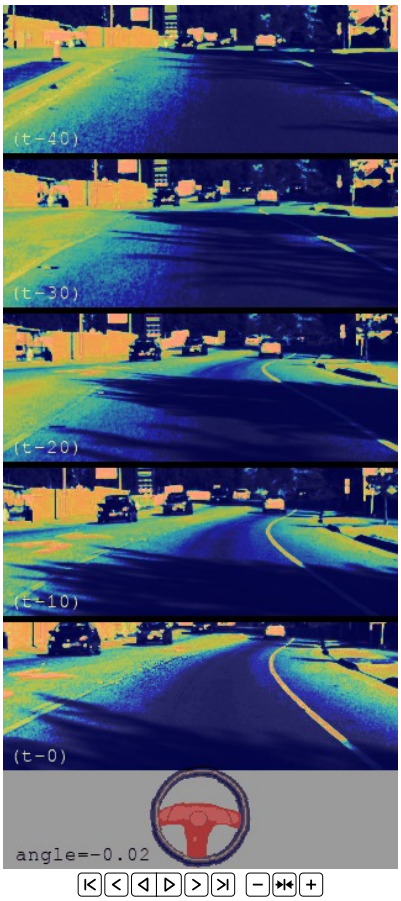


Figure 5.11.: CAM analysis of the Chauffeur-seq2seq model. The heatmaps are overlayed with the five input frames which have a temporal offset $(t - \Delta t)$ with t being the current time step.

By using different gradient modifiers we can generate CAMs for different purposes. Whereas a CAM generated with an unmodified gradient (Fig. 5.13a) shows pixels which support the correct prediction, the CAM generated with a negated gradient shows the parts of the image which would lead to a worse prediction (Fig. 5.13b), *i.e.* to sway the model from predicting the ground truth direction. Therefore, gradient modifiers can be used to investigate what improves a prediction or which image features in the input make the prediction worse. In the heatmaps we can

see that road markings mostly have a positive effect on the prediction, but it also occurs that attention on the center line of the road can make the prediction worse. In (b) we see that changes in the light conditions (*e.g.* a big area with a hard shadow on the road) have a big influence towards a worse prediction. For the lane following task the CAM generated with the negated gradient shows that attention on approaching cars does not help in the steering angle prediction, but rather makes it worse.

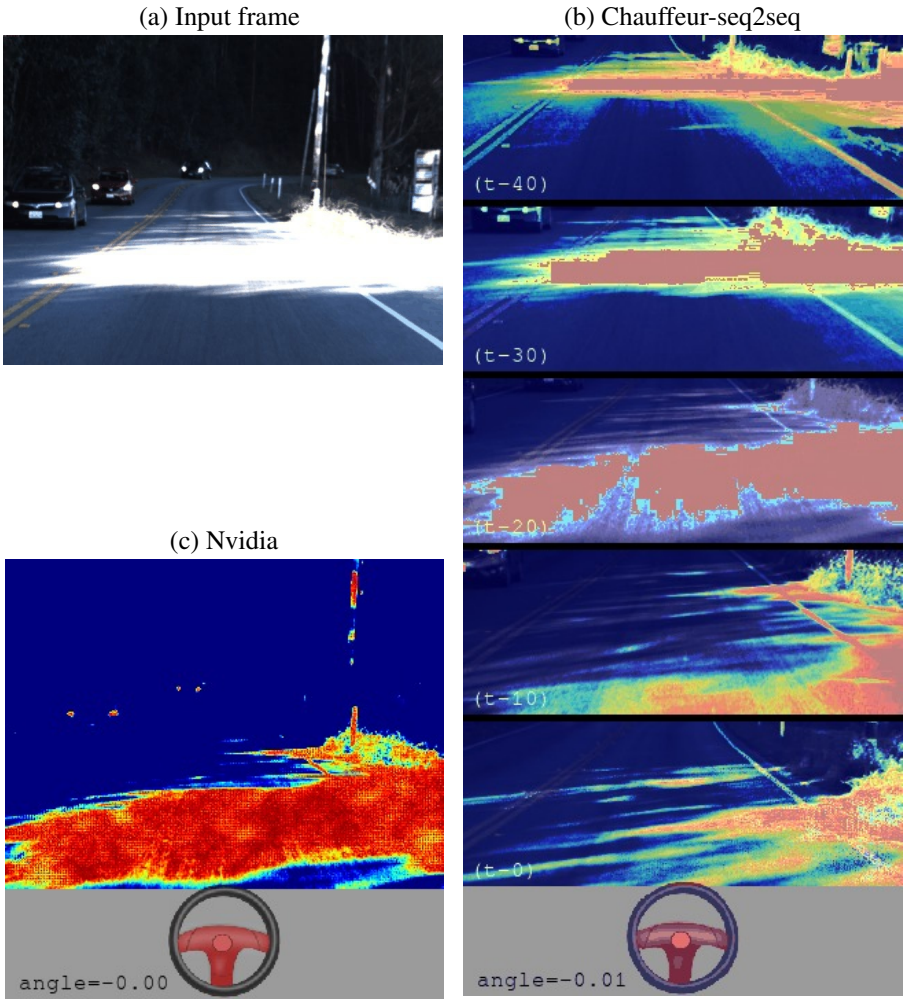


Figure 5.12.: Snippet from Fig. 5.10 and 5.11. Both models have high attention values for the change of shadow-light-shadow. The Nvidia model with only a single frame as input is likely to make a false prediction, whereas the Chauffeur-seq2seq model can use features from other input frames where there is still high attention at the road markings for driving straight.

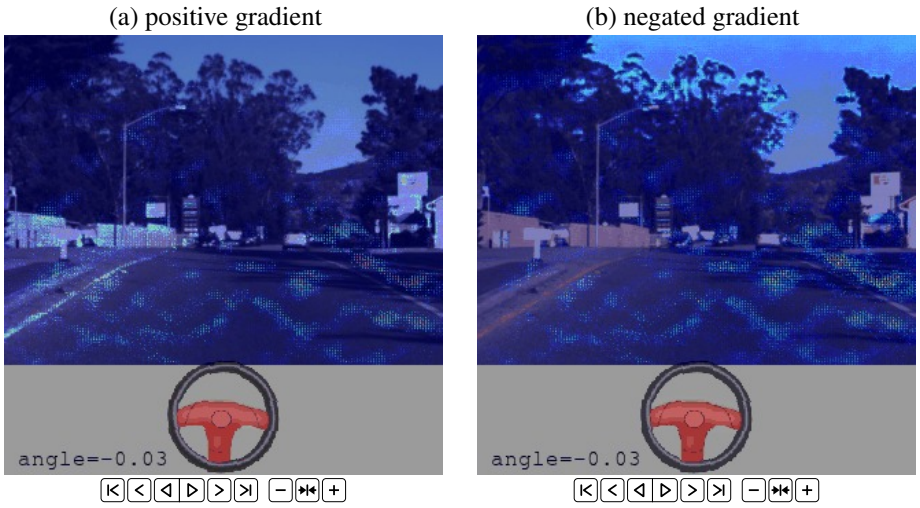


Figure 5.13.: CAM results of the Nvidia model [6], whereas the heatmaps with the negated gradient in (b) show image areas which would contribute to a worse prediction. The gradient is backpropagated from the final convolutional layer (see Fig. 3.12 for the architecture) to the input of the model conditioned on the output prediction, *i.e.* the steering angle.

Driving models trained on the BDD-V dataset

Next I use the BDD models [124] on the BDD-V validation set and look what triggers the predictions of the steering commands.

For the continuous BDD model we see in Fig. 5.14 that the model has strong attention to traffic lights. In this experiment we especially look at the speed predictions. In the video sequence the car first drives at a almost constant speed and stops then at a red traffic light in front of a crosswalk. Regarding the speed prediction we recognize that the model predicts a low speed or stopping especially in front of crossings, where we see that the CAM highlights the stripes of the crosswalks. This model has a strong attention on traffic lights. Through the entire video we can see how the model focuses its attention on the traffic lights as they give the essential information if a crossing can be passed or not. Additionally, the model recognizes road markings and also other cars are important for the model - including a focus on flashing brake lights. Interestingly, for pedestrians crossing the street the model only focuses on their feet and legs. One explanation could be that dark jackets are difficult to detect in a night scene.

Fig. 5.15 shows the same video sequence for the BDD-CNN and the FCN-LSTM model, which have both discrete outputs for “straight”, “stop or slow”, “left”, and “right”. The input video frames are overlayed with the generated heatmaps. The generation of the CAM is conditioned on the prediction of the model. That means we observe why a certain prediction is made. For both models we see in the beginning of the video,

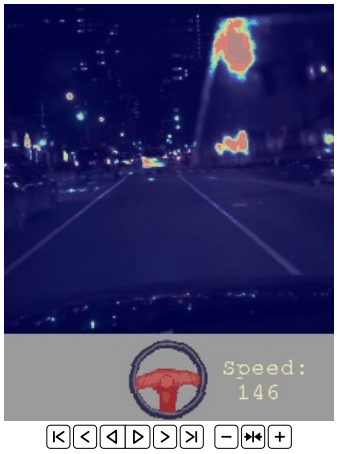


Figure 5.14.: CAM analysis of the continuous BDD model with overlayed input frames driving on a straight street with traffic lights. In the dashboard the predicted speed is displayed (class labels between 0 and 180 with 0 is ‘stop’ and 180 is driving fast).

that they concentrate on the lights of the other cars and the building in the distance. However the heatmaps show high attention values for reflections on the semi-transparent sticker on the windshield. This is bad, as this can potentially lead to wrong predictions. Furthermore, we notice that the models don’t pay too much attention to the straight center lines (*c.f.*, the models in Fig. 5.10). When the car approaches the right turn the BDD-CNN model predicts a left turn. This happens, because the model recognizes the diagonal hatching as road markings for a left turn. Five frames later the model recognizes its mistake and predicts the correct right turn. However, in a real-world scenario this could already have been too late and the car would have crashed into the oncoming traffic. The FCN-LSTM model has a similar heatmap, but it predicts to slow down or to stop. Also here we think the diagonal hatching causes confusion. in the end of the sequence both models show high attention for the brake light of the stopping car in front and they both correctly predict the “slow or stop” class.

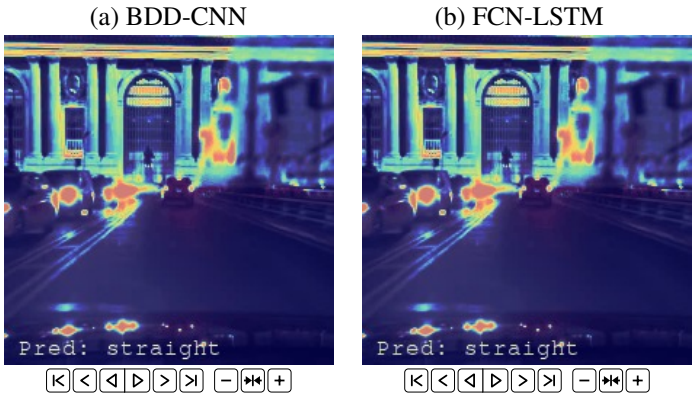


Figure 5.15.: CAM analysis of the BDD-CNN and the FCN-LSTM model with overlayed input frames.

5.3.3. Activation maximization without a learned prior

The generated images for maximized filters in the first convolutional layers of the Chauffeur-CNN are shown in Fig. 5.16. We see that for the first layer 'convolution2D_1' the generated images contain different colors and patterns, but they are not task specific for driving scenes. In the second convolutional layer 'convolution2D_2' we recognize that some filters learn to detect lines of various degrees, which fits to the driving task where a lot of road markings appear in the dataset. The other filter visualizations do not show such clear features, or at least they are not recognizable for a human. More high-level layers show more distinct line detectors.

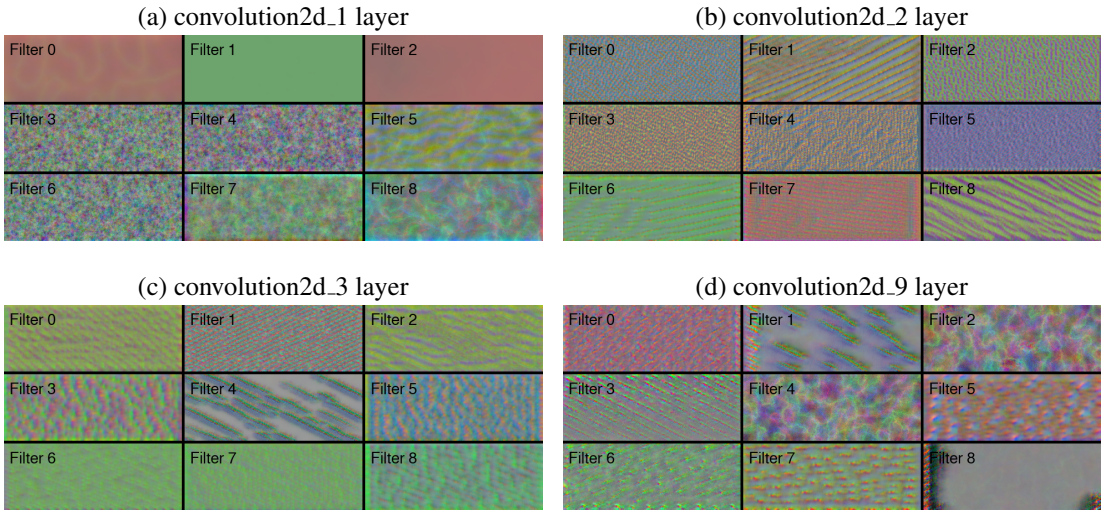


Figure 5.16.: Activation maximization of first convolutional layers from the Chauffeur-CNN.

The final layer visualization, Fig. 5.17, of the Chauffeur-CNN show how difficult it is to understand what the model has learned on the more abstract level. From the

deep learning perspective we know that the higher layers learn combinations of basic filter operations as shown in AM experiments for ImageNet classification [67, 77]. For the driving task however, without a strong image prior the AM algorithm does not generate images which are easily understandable for humans and they do not help in understanding the driving model.

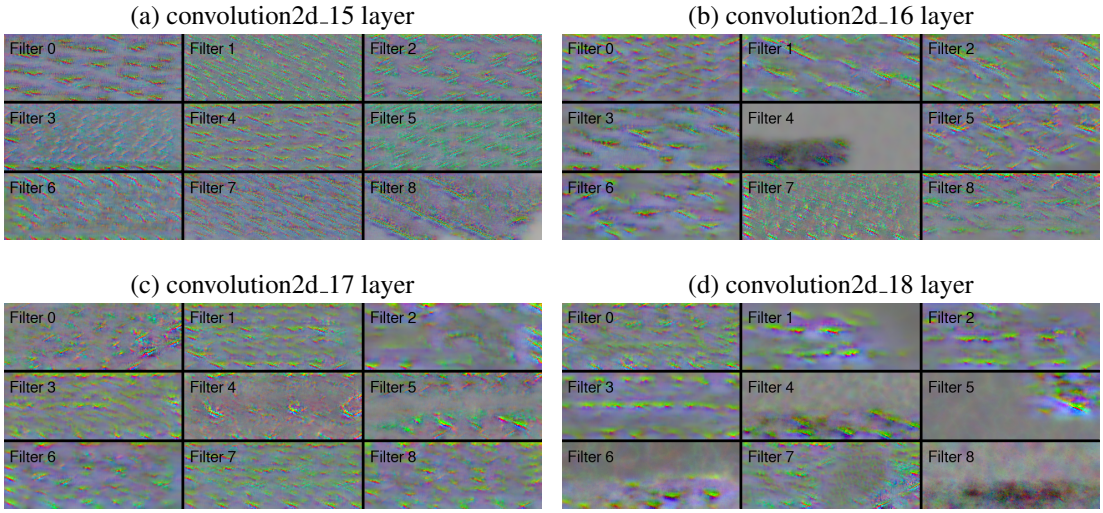


Figure 5.17.: Activation maximization of the final convolutional layers from the Chauffeur-CNN. Whereas the visualizations of the lower layers show line detectors for some units, here it is more difficult to interpret the generated images. However, we can recognize also that lines of various degrees appear in the generated images.

Next, I experiment to condition the AM by initializing non-randomly, *i.e.* I use an image from the driving dataset as starting point for the AM algorithm [77]. Fig. 5.18 shows the resulting AM visualization of the final convolutional layer of the Chauffeur-CNN. We recognize in some of the generated filter visualizations image features which look like road markings or roadsides showing a turn. This shows that it is possible to condition the AM by providing a prior in the initialization. However, it is still difficult to debug a driving model using this visualization as many generated images do not show features that are interpretable for humans.

We conclude that AM is a useful tool to find out what a neural network has learned, but it is still hard to interpret the results. Additionally, it is not easy to tune the parameters for the AM algorithm so that natural images are generated. Another problem is that the driving task has high temporal coherences and it is not possible to generate realistic video sequences using this framework. By using a deep generative network as prior the AM can be encouraged to generate natural images [74, 75]. We extend this concept of a learned prior and show results for the spatiotemporal domain in the next section.

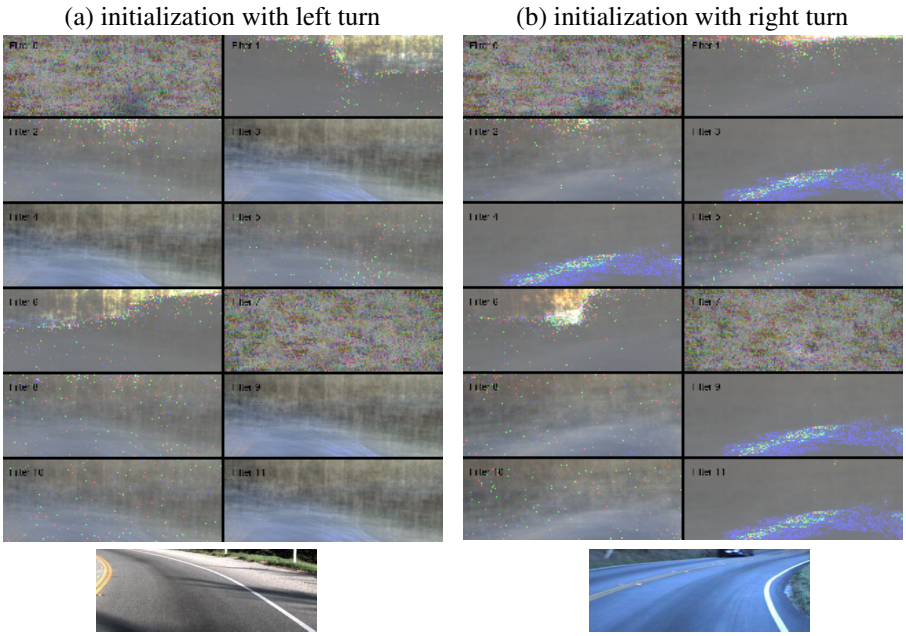


Figure 5.18.: AM at the final convolutional layer of the Chauffeur-CNN. In this experiment we use the shown images to initialize the AM algorithm. We see that the non-randomly initialization conditions the image generation and we can recognize a left/right turn in the filter visualizations.

5.3.4. ST-GAN: Short-term prediction experiments

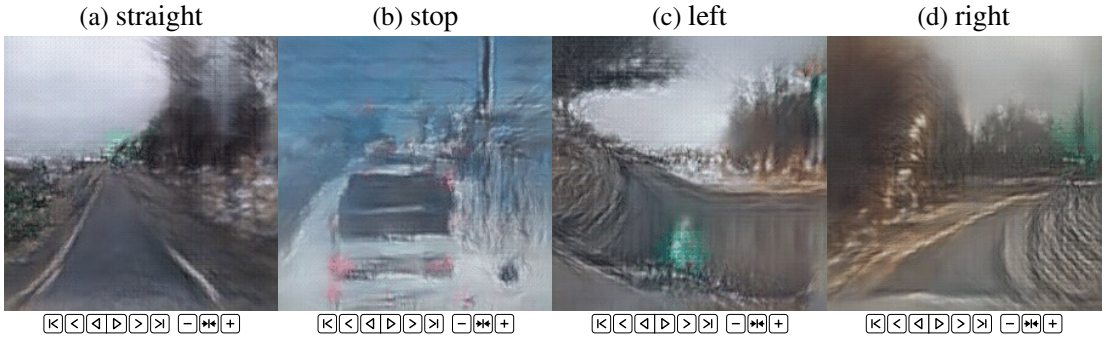


Figure 5.19.: Maximizing the activation of four output units of the discrete FCN-LSTM driving model [124] using ST-GAN.

Short-term video generation shows that the ST-GAN learns to model temporal coherences in a video sequence. The feature vector $\mathbf{x}$ is initialized with Gaussian noise: $\mathbf{x}^{(0:T-1)} = \mathcal{N}(0, 1)$. As condition model we use the discrete driving models of Xu *et al*. [124], which have four output units corresponding to driving commands: “straight”, “stop”, “left”, and “right”.

The first experiment shows generated sequences of 8 frames in Fig. 5.19, where each sequence maximizes the activation of the corresponding output unit. For the driving

command ”straight” the video shows a straight road. We can clearly see the road markings on the left side of the road on all frames. In the first frames we also notice the center line of the road. Beside the road we recognize the motion of driving through the scenery. This is not reflected in the textureless road structure. The video that maximizes the activation of the output unit “stop” shows a car which is first standing in front with flashing brake lights. After four frames the car in front accelerates and moves away from the camera. The sequence for “left” shows in the beginning a crossing with a left and right turn. This happens, because the dataset contains many crossings. The driving model learns this multi-modality, because without direction selects there are multiple correct directions when a crossing appears [124]. However, we can observe in the video, that the car follows the left turn. On the sides of the road we see that the car passes by trees. In the sequence for the command “right” we see a right turn in the road, where we can recognize the left road markings. In the first two frames of the all driving sequences with a lot of ego-motion (“straight”, “left, “right”) we recognize a glimpse of green light. This appears, because in these driving scenarios the car often passes by green traffic lights.

To clarify if a spatiotemporal generator is necessary to produce plausible videos, we compare video sequences generated with a spatial generator (S-GAN) and our spatiotemporal generator (ST-GAN). We maximize output units of the three discrete BDD models [124]: CNN (with a single frame as input), CNN-LSTM (an LSTM on top of the AlexNet [56]), and FCN-LSTM (an LSTM on top of a fully-convolutional CNN based on AlexNet [56]). The generated videos are generated using the short-term approach with a sequence length $T = 32$ frames and a total length of 96 frames. The top row in Fig. 5.20 shows that the S-GAN produces realistic frames for all driving commands, but the frames are temporally uncorrelated. For the CNN with the single frame as inut that is obvious as the prediction is based only on a single frame. However, the models (CNN-LSTM and FCN-LSTM) have recurrent layers, which should learn temporal structure in the data. It surprises that this is not happening. That might mean the LSTM is not suitable to learn spatiotemporal features. The bottom row in Fig. 5.20 shows the videos generated by using our ST-GAN. They show plausible sequences, where we can observe driving motion depending on the maximized driving command. ST-GAN can generate plausible driving sequences for all four driving commands and also for combinations of them, *e.g.* it is possible to generate a sequence driving straight and making a turn later. This is possible because the ST-GAN is trained on unlabeled data and the spatiotemporal filters also train on such combinations which are highly frequent in the dataset.

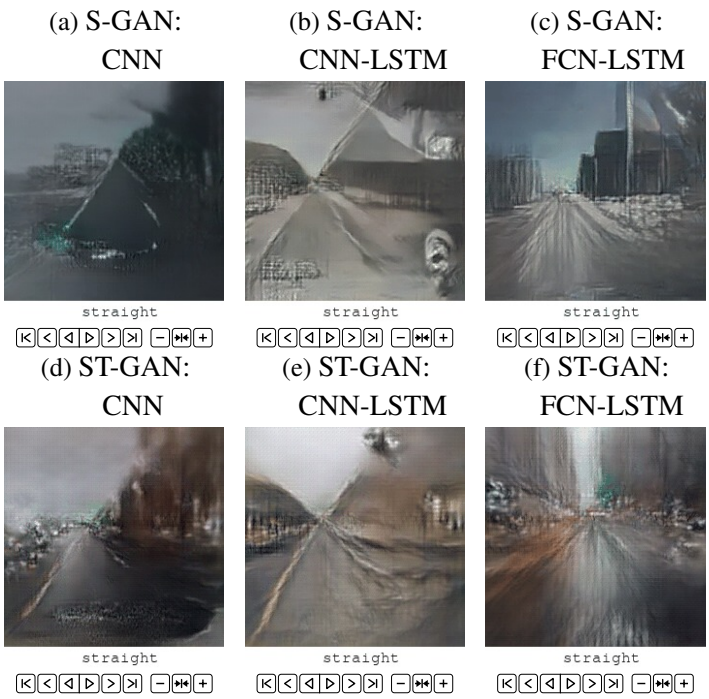


Figure 5.20.: Short-term visualization of the different discrete driving models. When using S-GAN, *i.e.* a purely spatial ConvNet, we observe that realistic frames are generated but without smoothness in time. While this is obvious for conditioning on a spatial (CNN) model, it is surprising for temporal sequence models (CNN-LSTM). Our ST-GAN (bottom) generates realistic videos for all 3 condition models.

5.3.5. ST-GAN: Long-term prediction experiments

To create a long video it is necessary to join video sequences. We show long driving sequences in Fig. 5.21, which show all the same track: “straight - left - straight - right - straight - stop”. In total one video has a duration of 96 frames. In this experiment the temporal convolution approach Ψ_{tconv} uses the overlap $\tau = 8$ frames (Fig. 5.4). The seq2seq approaches Ψ_{seq2seq} and Ψ'_{seq2seq} use a temporal depth of $\tau = 8$ frames.

We compare the short-time approach (with concatenated short-time sequences of length $T = 32$ frames) in Fig. 5.21a with our approaches for long-term video generation:

- the temporal convolution approach Ψ_{tconv} in Fig. 5.21b,
- the future-past conditioned seq2seq approach $\Psi_{seq2seq}$ in Fig. 5.21c,
- and the past conditioned seq2seq approach in Fig. 5.21d.

We observe that concatenation of generated clips in Fig. 5.21a produces scene cuts with strong appearance changes between the individual clips. In comparison Ψ_{tconv} in (b) produces temporally coherent video samples with smooth transitions in time. Also the seq2seq approaches Ψ_{seq2seq} and Ψ'_{seq2seq} in (c) and (d) respectively create temporally coherent long-term videos with diversity in scenery.

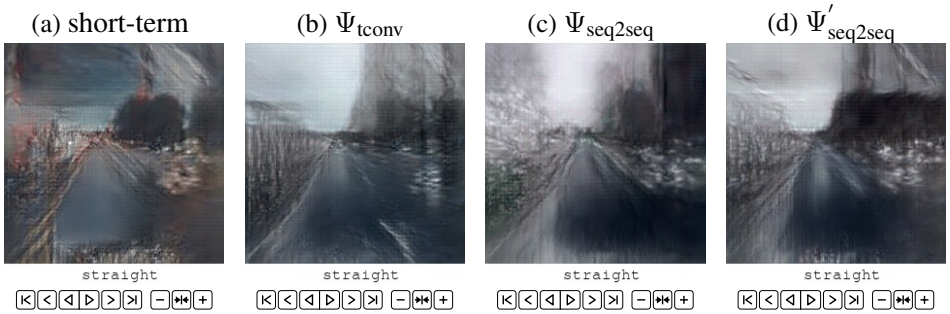


Figure 5.21.: Long-term generation of driving sequences by sequentially maximizing the prediction of units in C.

For closer inspection we show one transition for visually comparison in Fig. 5.22. We see a change from day to night within two frames in the video generated by the short-term approach. The temporal convolution and the seq2seq approach overcome this problem, because they condition the generated future sequence on the past. More transitions are shown in the driving videos in Fig. 5.21 every 32 frames for the short-term approach and every 16 frames for the long-term approaches. These driving videos are plotted unrolled in time in Appendix B.

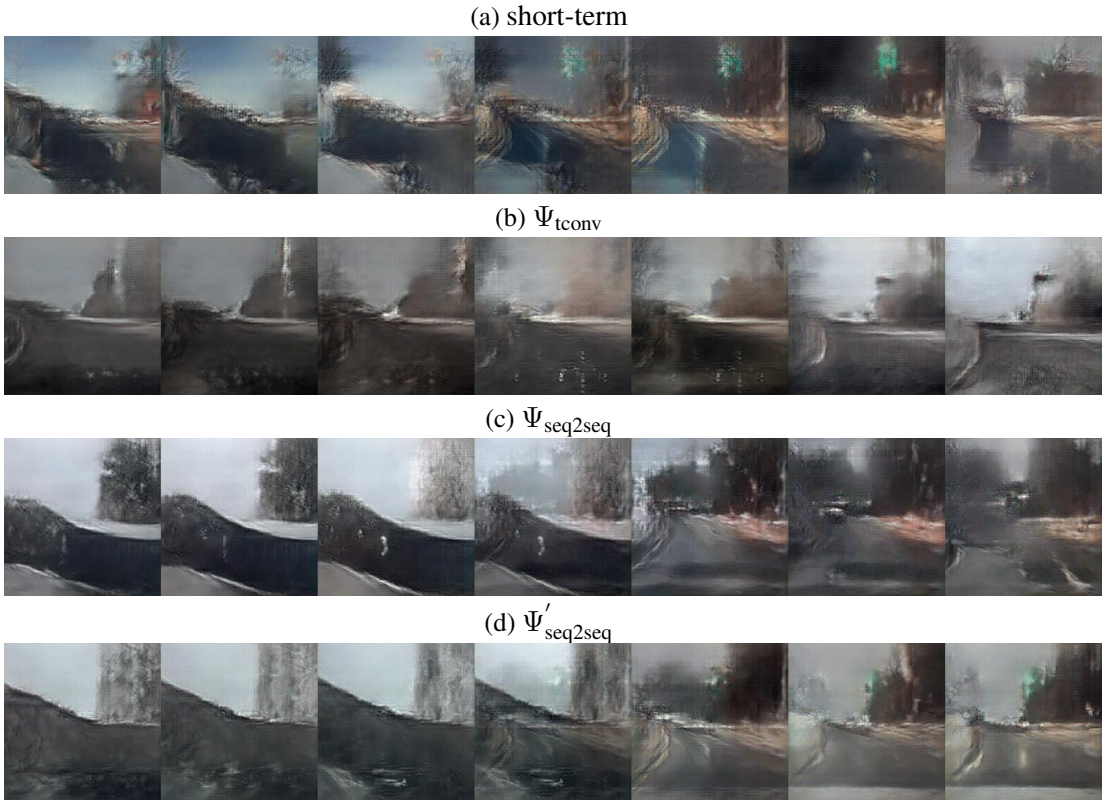


Figure 5.22.: Snippets of the driving sequences in Fig. 5.21. The video frames show the transition between two sequences maximizing a left turn, where the first three frames belong to the previously generated sequence. The short-term approach produces consecutive sequences that are visually dissimilar at the transition - a change from day to night is visible within two frames. The future prediction modules Ψ generate smooth transitions.

Whereas both of our approaches produce plausible long-term videos, the seq2seq approach convinces with performance. Using the temporal convolution approach the frames in the overlap τ are used in two sequences, *i.e.* in each further generates sequence we generate $T - \tau$ new frames. This does not apply to the seq2seq approach, where we always generate T new frames.

5.3.6. ST-GAN: Visualizing continuous driving models

In this section we show that the ST-GAN also works on continuous models, including models which are trained on a different driving dataset.

Instead of four driving commands the continuous Berkley DeepDrive model [124] with discretized bins offers a finer control of the steering angle. Additionally, we also experiment with the ResNet-50 [42] and the Nvidia model [6] (see Sect. 3.4.2), which have a regressive output layer and which are trained on the Udacity dataset [110] .

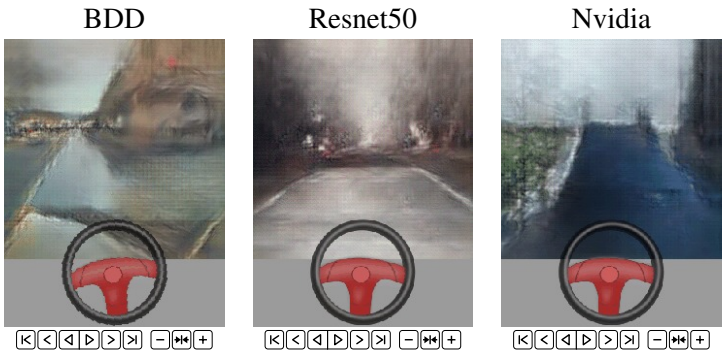


Figure 5.23.: Generating videos for continuous driving models using the activation maximization framework with our ST-GAN.

Figure 5.23 shows the generated videos using the ST-GAN. The steering wheel indicates the driving direction. All three models show that they have learned to recognize roadsides and outer road markings. For the Berkley model we recognize that red traffic lights and cars with flashing brake lights are generated in multiple frames. That happens, because we only maximize output neurons for steering directions and no speed information is given. Also the ResNet-50 shows one frame with a flashing brake light. Interestingly, the ResNet-50 and Nvidia models seem to only focus on near road structures *i.e.* on the bottom half of the visual field.

We notice that for the Berkley model (180 output neurons for driving directions), the center neuron (”driving straight”) shows the highest activation most time in the AM. Also two neurons (one for left, one for right) show a higher activation as all other surrounding neurons. The other neurons only reach low activations and are therefore not sufficiently maximized. Checking the ground truth in the dataset of the driving model, we recognize that for ≈50% of the annotations the center neuron fires. This implies a very unbalanced distribution of predicted steering angles and explains that the center neuron is stronger activated.

5.3.7. ST-GAN: Visualization of hidden units

Recurrent and fully-connected layers of the FCN-LSTM model

Additionally to the output layer, we also investigate what excites hidden units of the driving models. For the experiment we use the FCN-LSTM model [124] as condition model C.

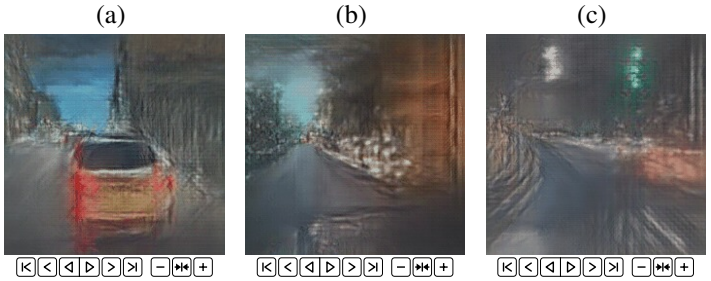


Figure 5.24.: Activation maximization of units in the LSTM layer of the FCN-LSTM driving model [124].

First, we maximize the activations of the LSTM layer of the driving model. Fig. 5.24 shows three samples of maximized units in the LSTM layer, where we can see in (a) a sequence of driving straight with a car in front which disappears after some frames and a red traffic light appears. The sequence ends with driving straight after the red traffic light vanishes. In (b) the video shows the stopping at a red traffic light and accelerating again in the last frames. Finally, the video in (c) starts with a left turn and then driving straight ahead, where we recognize a green light in the distance. Interestingly, the LSTM of the driving model does not learn to recognize temporal smoothness in videos (as explained in Sect. 5.3.4), but it learns combinations of driving commands, *e.g.*. "stop" - "straight" - "stop".

Next, we show results for maximizing the hidden neurons in layer fc7 which have a spatial resolution of 7×7 (for an 227×227 sized network input). Example sequences are illustrated in Fig. 5.25. Beside braking cars in front for the 'car' cluster, we see driving scenes on roads in daylight or in the night, and stopping at crosswalks where we notice green and red traffic lights and sometimes we also notice people crossing the street.

We also cluster the generated sequences of layer fc7 based on their appearance and analyze the distribution of the clusters in Table 5.3. We notice that only few units can be maximized sufficiently; that means, the activation of the unit does not exceed 1% probability (computed with softmax over all units in the layer). 1.5% of the generated sequences show dark sequences and only 0.7% show videos in daylight. 97.8% of the units cannot be explicitly maximized, which indicates lots of 'dead' neurons. That might mean that the architecture and number of neurons in the layer is not optimal for this



Figure 5.25.: Activation maximization of units in the fc7 layer of the FCN-LSTM driving model [124].

driving task. It is still surprising to find these 'dead' neurons, because the ST-GAN and the FCN-LSTM driving model have been trained on the same data and should therefore be able to activate these neurons.

Table 5.3.: Activation maximization of **layer fc7** of the FCN-LSTM driving model [124]. This evaluation shows the distribution over distinct scenes which are generated by maximizing different neurons in this layer (#neurons=4096).

fc7 layer	car	street	crosswalk
night scene	0.3%	1.0%	0.2%
daylight scene	0.1%	0.4%	0.2%
not categorizable	0.0%		
not maximized	97.8%		

Next, we explore visualizations at layer 'fc6', which is the first fully connected layer in the driving model [124]. In Fig. 5.26 we show examples of selected neurons which represent clusters of scene descriptions in daylight and night. The captions of the videos correspond to the scene descriptions in Table 5.4. In 'night: car' we see a stopped car close in front of our windshield and a read traffic light above the car. In the night scene of 'street' we notice street markings, a car appearing in the distance for a short moment. For 'night: crosswalk' we observe people crossing the street on the crosswalk while we also see green traffic lights. In the daylight scenes blue sky attracts attention. For the 'car' scene we see remark the flashing brake light of the car in front. For 'daylight: street' we recognize cars in the distance and buildings on the side of the street. The crosswalk in daylight additionally shows a red traffic light.



Figure 5.26.: Visualization of hidden neurons of C in layer fc6.

Table 5.4.: Activation maximization of **layer fc6** of the FCN-LSTM driving model [124]. This evaluation shows the distribution over distinct scenes which are generated by maximizing different neurons in this layer (#neurons=4096).

fc6 layer	car	street	crosswalk
night scene	6.1%	31.0%	6.5%
daylight scene	5.4%	22.8%	4.4%
not categorizable	11.4%		
not maximized	12.4%		

We divide the generated videos for maximizing the hidden layer 'fc6' into eight clusters based on their appearance. The distribution of the clusters is listed in Table 5.4. We find that approximately 43.6% of the 4096 neurons produce dark images, *i.e.* night scenes, as preferred input, whereas only 32.6% of the neurons fire for daylight videos. 12.4% of the units cannot be explicitly maximized, and 11.4% produce videos that show mixtures of the six scenes and cannot be assigned to a single cluster. Compared to the layer fc7, where most neurons cannot be sufficiently maximized, in this layer only 12.4% of units exist which activation does not exceed the threshold of 1% probability.

Visualizing hidden convolutional layers of the FCN-LSTM model

We can also show visualizations of convolutional layers. Fig. 5.27 shows preferred input sequences for selected 'conv5' filters. While some filters have learned to perceive turns, other filters recognize elements in driving scenes like crosswalks or traffic lights. There are also filters which show high activations for cars with or without flashing brake lights.

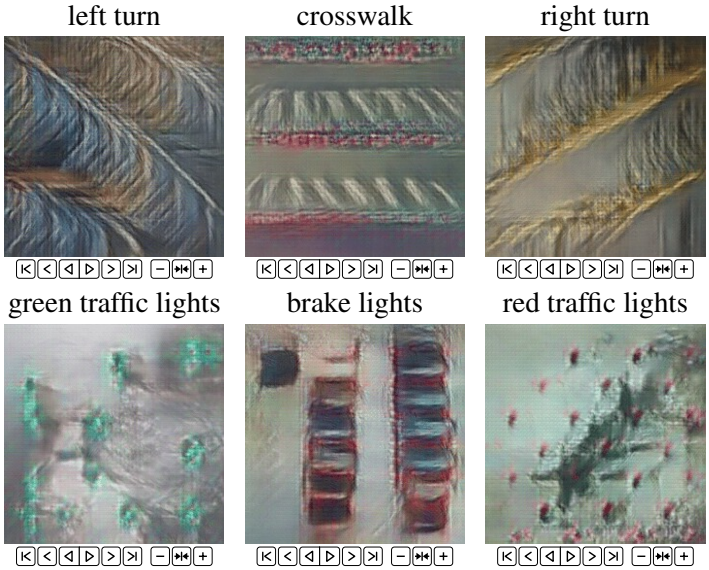


Figure 5.27.: Spatiotemporal AM of conv5-filters conditioned on the FCN-LSTM driving model [124].

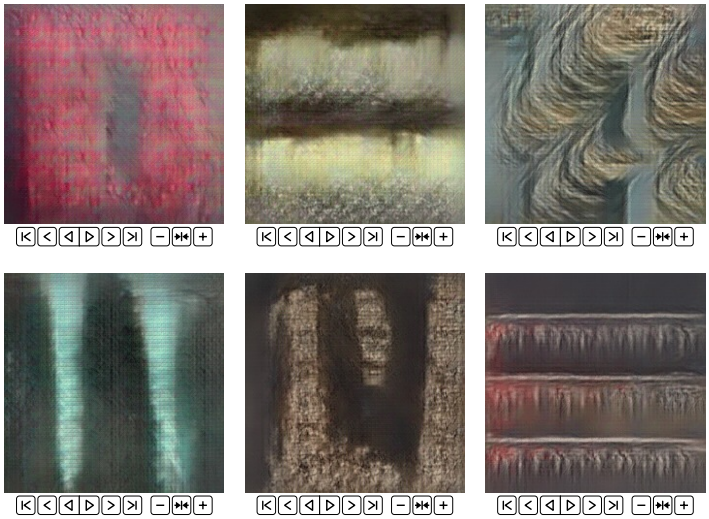


Figure 5.28.: Spatiotemporal AM of conv4-filters conditioned on the FCN-LSTM driving model [124].

Fig. 5.28 contains sequences maximized on a unit of layer 'conv4'. The generated video sequences show already less similarities to driving data. The 'conv3' layer visualizations in Fig. 5.29 already shows almost no similarity to driving sequences. However, we find filters that prefer lines of a certain degree as input, whereas other filters have high activations on motion. Furthermore, Fig. 5.30 shows three samples of the spatiotemporal AM results for layer 'conv2' which show simple textures and structures with edges. We note that our ST-GAN produces sequences with spatiotemporal structure although the visualized filters are spatial.

Appendix B shows additional sample visualizations for units at all these layers.

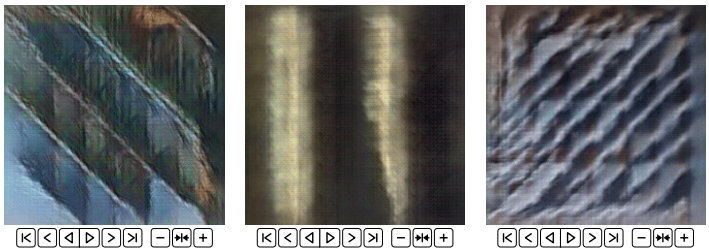


Figure 5.29.: Spatiotemporal AM of conv3-filters conditioned on the FCN-LSTM driving model [124].

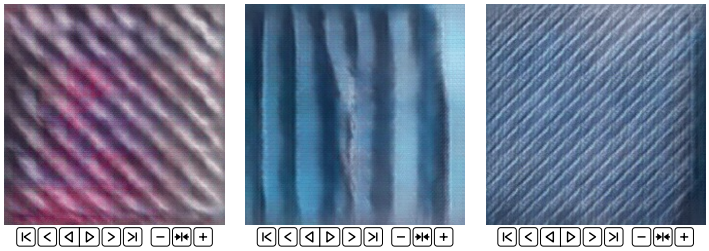


Figure 5.30.: Spatiotemporal AM of conv2-filters conditioned on the FCN-LSTM driving model [124].

5.3.8. ST-GAN: Conditioning on multiple models

Additionally to the driving model that we want to visualize, we can also condition the video generation on other neural networks in parallel. As shown in Figure 5.31 we add the scene model C_{scene}, which is an AlexNet trained to classify different places from the MIT places dataset [134]; see Fig. 5.33 for samples of the dataset.

The AM algorithm only changes slightly as the scene model is just another conditional model like the driving model. Thus, we add the gradient term for the scene model, weighted by $\epsilon_{c,\text{scene}}$, to update equation (5.22):

$$\begin{aligned}\Delta \mathbf{x}^{(t:t+T)} &= \epsilon_{\text{AE}}(\mathbf{x}_{\text{AE}}^{(t:t+T)} - \mathbf{x}^{(t:t+T)}) + \mathcal{N}(0, \epsilon_{\text{noise}}^2) \\ &+ \epsilon_{\Psi}(\mathbf{x}_{\Psi}^{(t:t+T)} - \mathbf{x}^{(t:t+T)}) \odot \sigma(T) \\ &+ \epsilon_c \frac{\partial \log C_c(G_{ST}(\mathbf{x}^{(t:t+T)}))}{\partial G_{ST}(\mathbf{x}^{(t:t+T)})} \frac{\partial G_{ST}(\mathbf{x}^{(t:t+T)})}{\partial \mathbf{x}^{(t:t+T)}} \\ &+ \epsilon_{c,\text{scene}} \frac{\partial \log C_{c,\text{scene}}(G_{ST}(\mathbf{x}^{(t:t+T)}))}{\partial G_{ST}(\mathbf{x}^{(t:t+T)})} \frac{\partial G_{ST}(\mathbf{x}^{(t:t+T)})}{\partial \mathbf{x}^{(t:t+T)}},\end{aligned} \tag{5.24}$$

where the parameter $\epsilon_{c,\text{scene}}$ controls how much the video generation is influenced by the scene model.

We use the discrete FCN-LSTM model as shown in Fig. 3.18a to generate a driving video. In this experiment we simultaneously maximize the activation of a scene at the output of the scene model C_{scene} and a driving command at the output of the driving

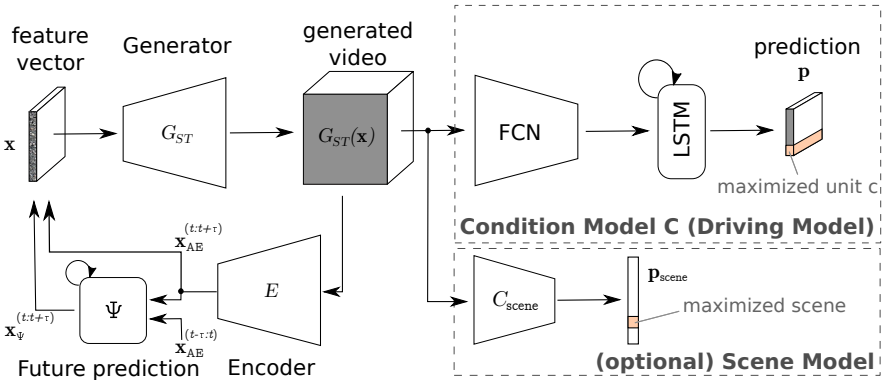


Figure 5.31.: Generating a video conditioned on two models: a driving model a the scene model. We use the FCN-LSTM driving model of [124] and an AlexNet trained on the MIT places dataset [134] to change the driving scenery.

model C. The scene model C_{scene} works in the image-domain and changes the scenery of the generated video - it does not supply motion information (our ST-GAN is responsible for spatiotemporal coherence).

Adding a scene model adds more diversity to the generated videos. Figure 5.33 shows the outcomes of generating videos by maximizing different output units of the scene model. In 'forest path' we can see trees at the sides of the road, whereas windmills appear beside the road when we maximize the respective class. Interestingly, it is possible to generate videos in winter scenes by maximizing the class 'ski resort' although the



Figure 5.32.: Dataset samples of the MIT places dataset [133]. The scene model C_{scene} is trained on this dataset [134]. Please note that we do not use these images to generate the videos in Fig. 5.33.

ST-GAN has never seen snow before. We can also generate sequences for driving through fields (*e.g.* class 'wheat field') or over golf course turf. Maximizing the place 'supermarket' gives the impression of driving through supermarket shelfs. Maximizing the class 'desert sand' colors the side of the road like yellow sand; whereas the scenery changes in the end of the video, where we recognize dark sand with wavelike pattern. In the sequence maximizing the scene 'cathedral' we pass by salient buildings with towers. In the underwater scene we notice a blue cast and vegetation at the side of the "road" which could be an illustration of corals.



Figure 5.33.: Generated videos conditioned on the FCN-LSTM driving model C [134] and the scene model C_{scene} trained on the MIT places dataset [134]. A sequence of units (driving commands) is maximized at the output layer of the driving model, and a scene is maximized at the output layer of the scene model.

5.3.9. ST-GAN: Generating driving videos from an image

Instead of initializing $\mathbf{x}$ with Gaussian noise it is also possible to use the encoded feature vector from a real or artificial image: $\mathbf{x}_0^{(i)} = E(\mathbf{y}_{\text{init}}), i \in \mathbb{N}_0, \forall i < T$.

Figure 5.34 shows the activation maximization of a driving sequence, where all feature vectors were initialized with the encoded daylight image of a highway or a blue-gray drawing. The images for the initialization are shown downscaled beside the generated video. We use the discrete FCN-LSTM driving model, a sequence length of $T = 32$ frames and the seq2seq approach for smoothing the transitions. The videos have a total length of 64 frames.

The results show that qualitatively good videos are generated although the initialization with images does not contain motion information. Using the highway image we successfully condition the AM to generate a video with buildings on the side and blue sky. Although the image shows a straight road, the generated video effectively maximizes turns, whereas the turn to the right is more distinct. The experiment with the blue-gray image shows that we can "condition" the algorithm to generate driving videos with blue sky without presenting real-world data.

This experiment shows that the scenery of the driving videos generated with our ST-GAN can be affected by initializing with non-random data. This technique could be used for augmentation of training data for driving models.

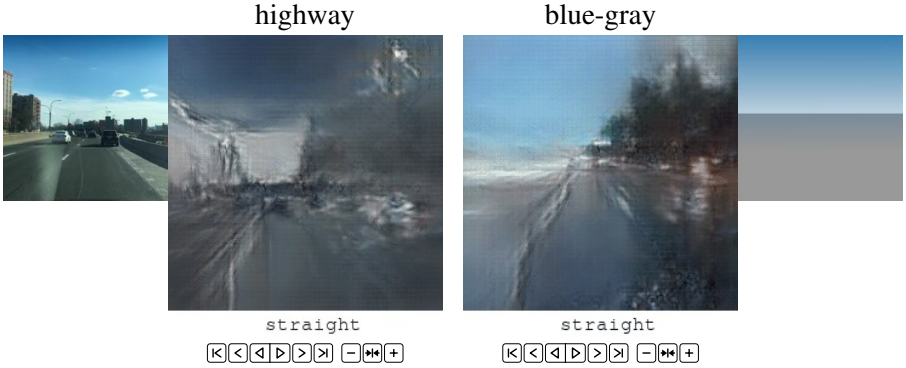


Figure 5.34.: Sequences maximizing the activation of the output layer of the discrete FCN-LSTM driving model [124]. The feature vectors $\mathbf{x}$ are initialized with encoded images (downscaled images beside). If we initialize with the highway image we recognize that the buildings on the left side also appear in the generated video for left turns and driving straight ahead, whereas they disappear in the right turn. The video initialized with the blue-gray drawing also contains some buildings or trees, but they are not as distinctive. However, for both videos we notice a blue sky like appearance as in the images used for initialization.

5.4. Conclusion

Visualizing and understanding deep driving models is essential before autonomous cars can be shipped at large scale.

We see in the t-SNE analysis that the CNN was successfully trained on the driving task as it learned useful features that group steering angles with similar values in a three-dimensional representation.

The experiments with class activation maps show why models fail to predict a correct steering angle, *e.g.* they show that the ResNet-50 model focuses strongly on the horizon line and the sky. This indicates that the model overfits on the training set, *i.e.* it learned the noise in the data. However, the heatmaps also show that the models focus on road markings and roadsides, which are also important features perceived by a human driver. A drawback of class activation maps is that they are only as good as the input data for the evaluation, *e.g.* they cannot show if the model has learned to recognize crosswalks if the test data does not contain frames with crosswalks. They also do not visualize if the model has learned a spatiotemporal representation of the driving task.

Generating preferred input for a driving model using AM can show what features the model has learned. However, performing AM without a learned prior as shown in Sect. 5.3.3 produces barely natural images and the selection of hyperparameters is difficult. The few natural images that are generated are not easy to understand by the evaluator. Having already troubles in the image generations, I think that the more complex video generation would work even worse.

Therefore, I present the the ST-GAN, which is trained for generative video synthesis from a large-scale unlabeled video dataset with driving sequences. The approach shows that an autoencoder coupled with a future prediction module enables the ST-GAN to produce videos of arbitrary length. Applied in the activation maximization framework the ST-GAN generates long-term videos which are natural, realistic, in high-resolution and easily understandable for humans. These videos show the preferred input for the condition model – the deep driving model – and provides insight what the network has learned during the training procedure. It reveals strengths but also weaknesses of the driving model and has the potential to help in improving and further developing the analyzed model.

Video generation is still an ongoing research task, but with ST-GAN it is now possible to provide first insights how deep spatiotemporal models work.

6. Summary and Outlook

This chapter summarizes the research and findings presented in this thesis. The contributions of the thesis are twofold: 1) the improvement of driving models (Chapter 4), and 2) understanding and visualizing of driving models (Chapter 5). First, for improving driving models I have presented models with a sequence-to-sequence (seq2seq) architecture, which predict the current and future steering commands. The future predictions of previous time steps are used to improve the current prediction of the steering angle. Second, for understanding driving networks I have highlighted the salient pixels in the input using class activation maps (CAM). For being independent from testing data, I have implemented algorithms based on activation maximization (AM). In particular I have proposed a spatiotemporal GAN for video generation conditioned on maximally activated units in the driving model.

Improving driving models: The experiments in Chapter 4 confirm that heavy data augmentation is necessary for training deep neural networks, like ResNet-50 [42], on a small driving dataset like Udacity to prevent overfitting.

The evaluation of the future predictions shows that the error rises with increasing range of the prediction, *i.e*. predictions that lie farther in the future are less reliable. These reliability scores are used to adaptively superimpose future predictions to improve the current steering prediction. Although this adaptive method depends on the test errors, and therefore can be seen as cherry picking, it contributes to the idea that a non-linear mapping from future predictions to a current superimposed steering command is possible. By training a feed-forward neural network on this non-linear mapping, I show that the superposition method reduces the steering error that the model makes without using test data.

Interesting directions for future research would be to incorporate navigation information, *e.g*. from a GPS, into the driving model. This would enable the car to take turns at crossings or to leave the highway at an exit. Furthermore, testing the algorithm in a real car or in a simulator could give more insights of the performance of the model, because the evaluation metric of the Udacity dataset does not consider consequences of bad predictions.

Visualizing driving models: The experiments in Chapter 5 show that CAMs can be used to evaluate if a model overfits, *e.g.* on the horizon line, or why it makes a bad predictions, *e.g.* highlighting a diagonal hatching which is interpreted as center line of the road. Although in many cases the computed heatmaps are easy to interpret, it is not always clear how and why the highlighted features contribute to the prediction. Furthermore, the quality of the evaluation always depends on the available data.

To generate the preferred input of a driving model, I have proposed a spatiotemporal GAN (ST-GAN) integrated in an AM framework. This model can produce driving videos which maximally activate the output neurons which represent driving commands. The ST-GAN generates videos of high resolution, which show realistic motion and natural video frames. Another contribution is the introduction of methods for long-term video generation with the ST-GAN. These methods enable the model to generate driving videos of arbitrary length. By conditioning the video generation on a second condition model, which is trained on scenes, we show that the scenery of the video can be changed by maximizing different scene units. This technique could be used as augmentation technique for improving the training of driving models.

With ST-GAN also the hidden layers of the driving model can be visualized. Interestingly, the fully-connected layers contain units which cannot be maximized explicitly. This could mean, that these neurons are 'dead', *i.e.* they never got a gradient in the training nor will they play a role in the prediction. Notably, the high-level fc7 layer has a much larger fraction of these 'dead' neurons than the lower layer fc6. This finding indicates that the architecture of the driving models might not be optimal for the task. This shows another application for the ST-GAN to analyze and understand driving models.

For future research it would be interesting to reduce the motion blur which appears in the driving videos generated with the ST-GAN. Furthermore, testing and evaluating the ST-GAN on other datsets, *e.g.* the UFC101 dataset [98], could show if the model is applicable to other tasks.

Appendix

A.

More hidden layer visualizations using ST-GAN

In this appendix we present more samples of the hidden layers visualizations from Sect. 5.3.7:

- LSTM layer: Fig. A.1
- fc7 layer: Fig. A.2 and A.3
- fc6 layer: Fig. A.4 and A.5
- conv5 layer: Fig. A.6
- conv4 layer: Fig. A.7
- conv3 layer: Fig. A.8
- conv2 layer: Fig. A.9



Figure A.1.: More visualizations for the **LSTM** layer. Each sequence is generated by maximizing a different unit in the layer.

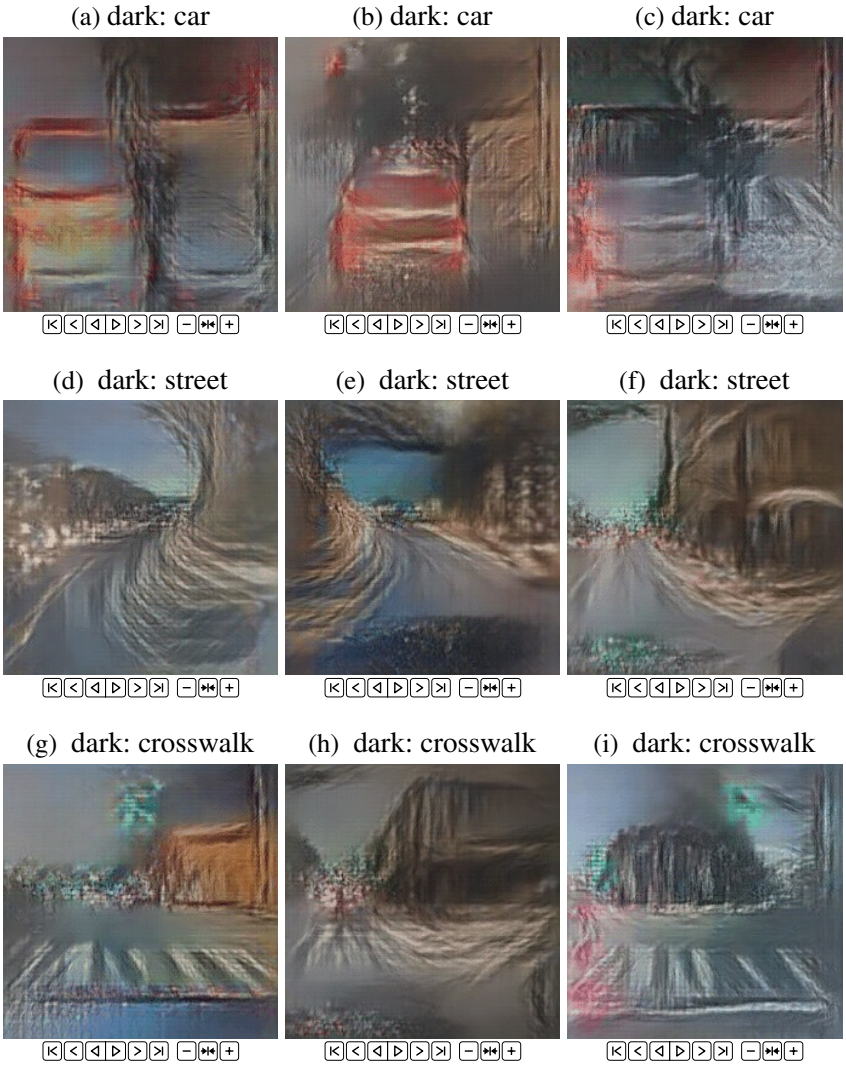


Figure A.2.: More **fc7** visualizations for dark scenes. Each sequence is generated by maximizing a different unit in the layer.

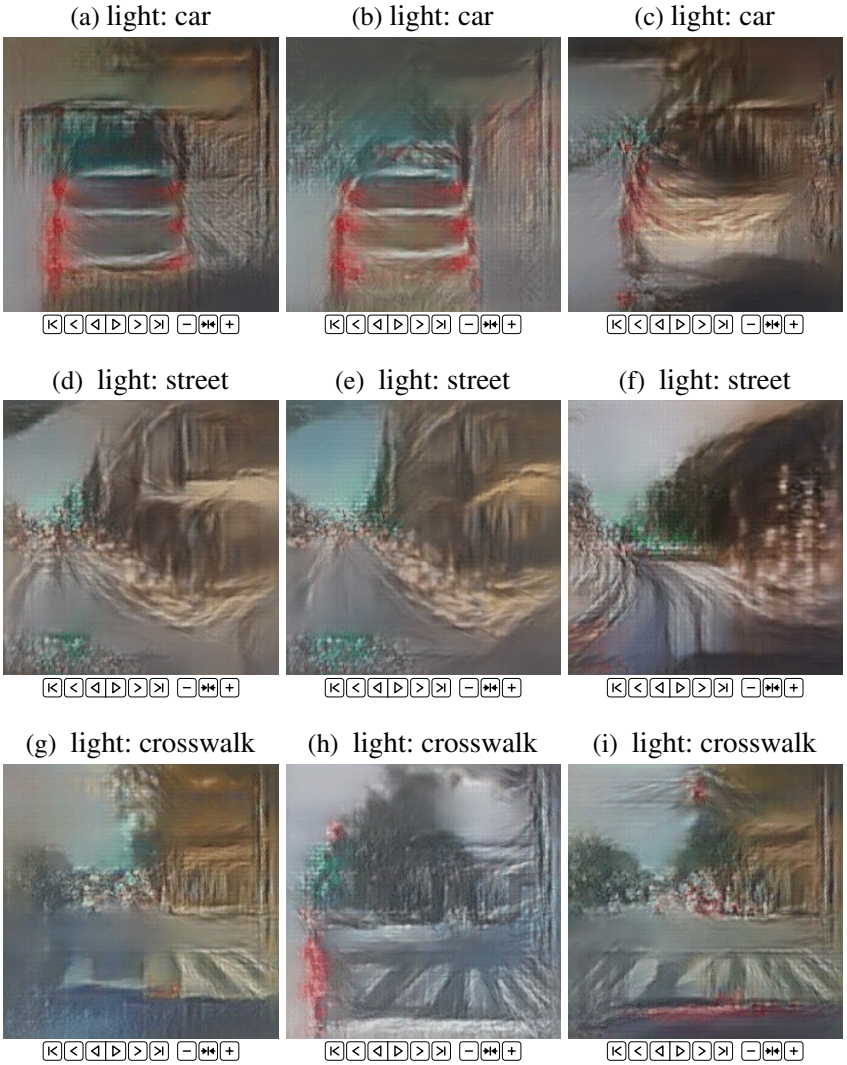


Figure A.3.: More **fc7** visualizations for daylight scenes. Each sequence is generated by maximizing a different hidden unit.

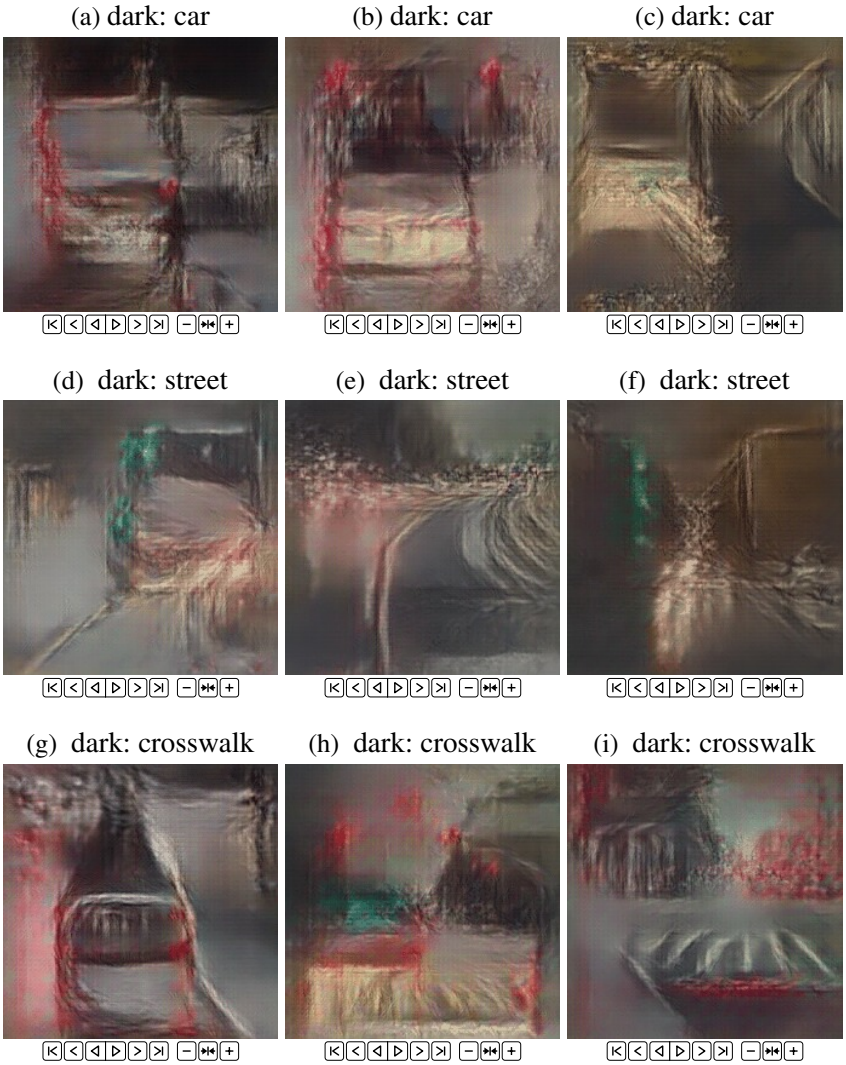


Figure A.4.: More **fc6** visualizations for dark scenes. Each sequence is generated by maximizing a different unit in the layer.

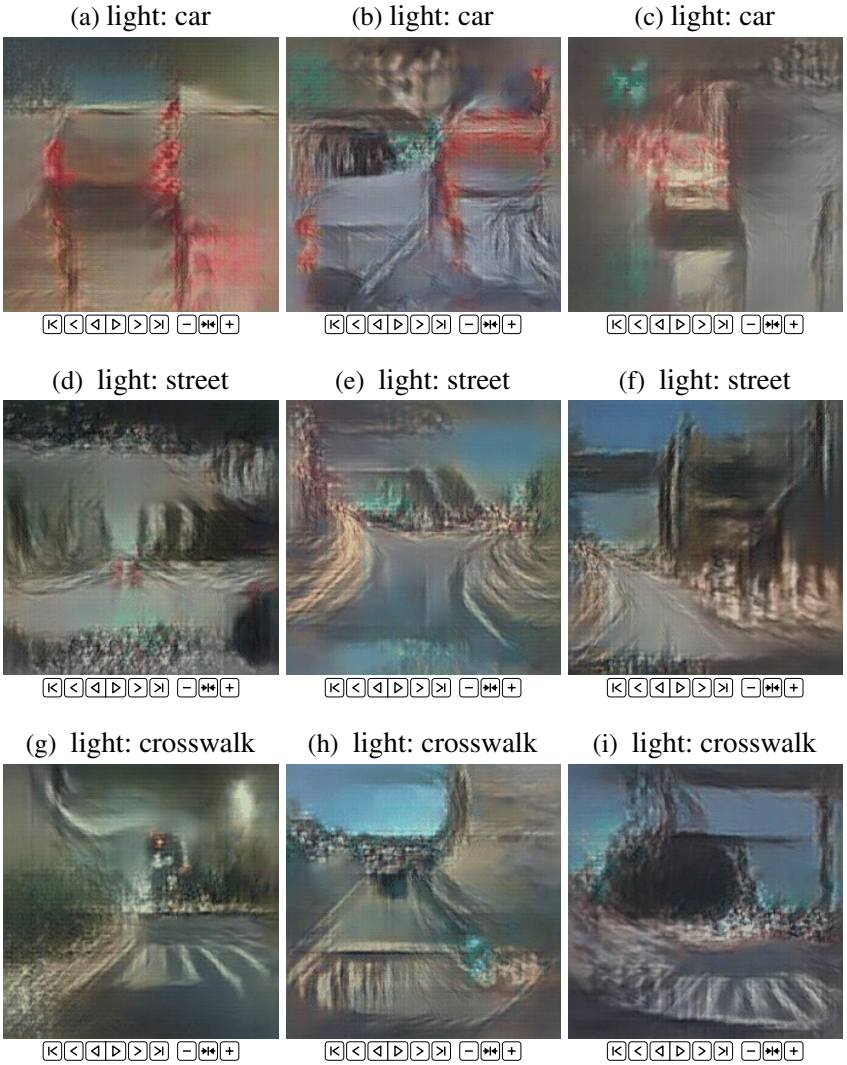


Figure A.5.: More **fc6** visualizations for daylight scenes. Each sequence is generated by maximizing a different hidden unit.

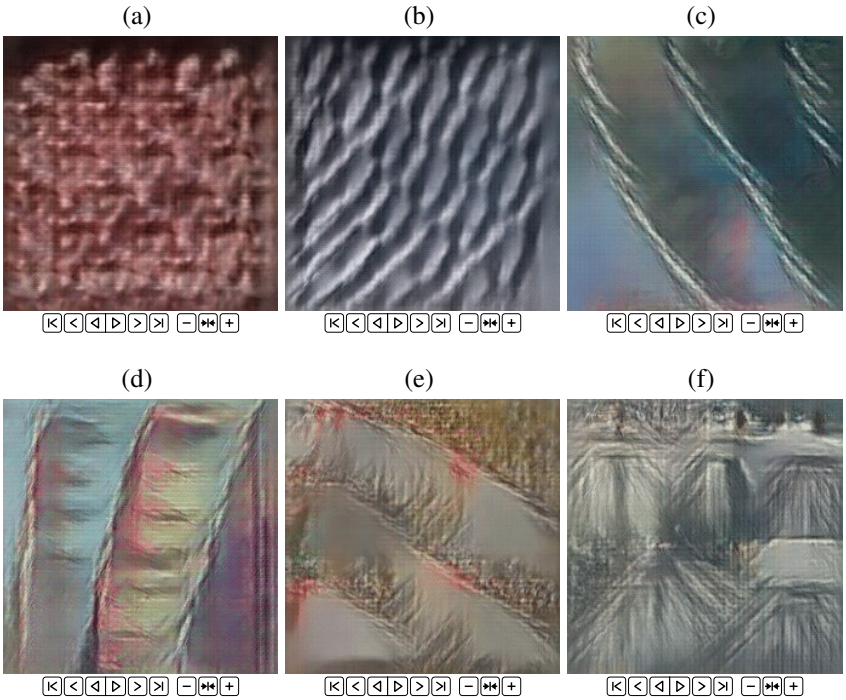


Figure A.6.: More **conv5** filter visualizations. Each sequence is generated by maximizing a different unit in the layer.

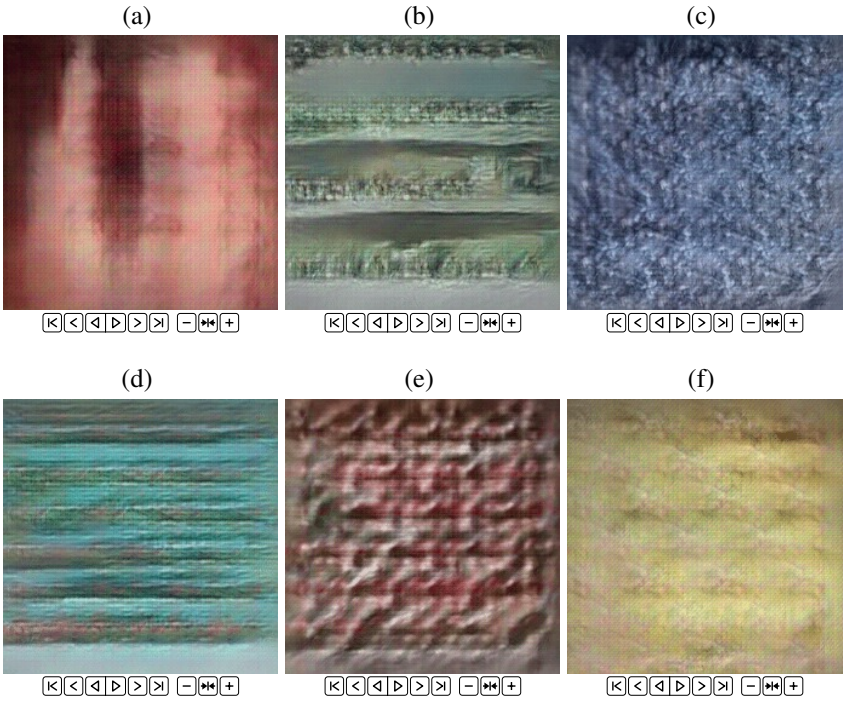


Figure A.7.: More **conv4** filter visualizations. Each sequence is generated by maximizing a different unit in the layer.

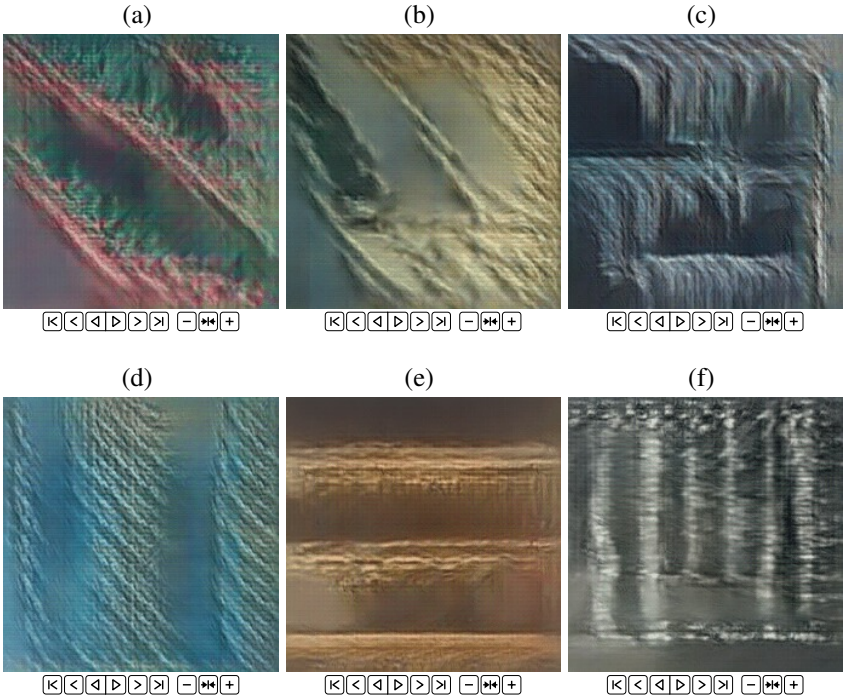


Figure A.8.: More **conv3** filter visualizations. Each sequence is generated by maximizing a different unit in the layer.

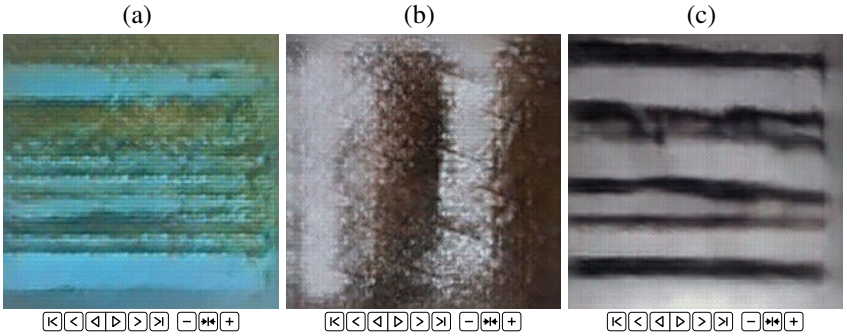


Figure A.9.: More **conv2** filter visualizations. Each sequence is generated by maximizing a different unit in the layer.

B.

Unrolled video clips

Sect. 5.3.4 to 5.3.9 contain multiple experiments where we generate video sequences. In this appendix we plot the video sequences unrolled in time for closer inspection. Table B.1 lists all video sequences that we plot unrolled in time in this thesis.

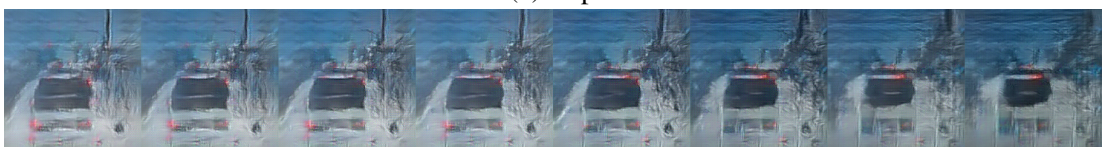
Table B.1.: This table lists the experiments and the references to the generated video sequences as well as the video frames unrolled in time.

Sect.	**Experiment**	**Figure**	
		video	unrolled in time
5.3.4	four directions	Fig. 5.19	Fig. B.1
5.3.4	discrete models	Fig. 5.20	Fig. B.2 to B.7
5.3.5	short-term	Fig. 5.21a	Fig. B.8
5.3.5	Ψ_{tconv}	Fig. 5.21b	Fig. B.9
5.3.5	Ψ_{seq2seq}	Fig. 5.21c	Fig. B.10
5.3.5	Ψ'_{seq2seq}	Fig. 5.21d	Fig. B.11
5.3.6	continuous models	Fig. 5.23	Fig. B.12 to B.14
5.3.7	hidden layer	Fig. 5.26	Fig. B.15 to B.16
5.3.8	scenery change	Fig. 5.33	Fig. B.17 to B.21
5.3.9	initialization with image	Fig. 5.34	Fig. B.22 to B.23

(a) straight



(b) stop



(c) left



(d) right



Figure B.1.: Video frames unrolled in time showing the videos from Fig. 5.19. The video generation is conditioned on the discrete FCN-LSTM driving model [124]. We see short video sequences, where the output unit for a driving command is maximized. The video with the maximized "straight" unit shows a straight road structure, where we recognize the focus of expansion motion of driving through the scenery (which is not reflected in the textureless road structure). The video that maximizes the activation of the output unit "stop" illustrates a car which is first standing in front with flashing brake lights and after four frames that car accelerates from the observer and the red brake lights disappear. The sequence for "left" and "right" show turns of the road. Interestingly, we see a green light flashing in the first two frames of all sequences with forward motion but not in the "stop" case. This is reasonable as in these driving scenarios the car often passes by green traffic lights.



Figure B.2.: Video frames unrolled in time of Fig. 5.20 for S-GAN: CNN. When using S-GAN, *i.e.* a purely spatial ConvNet, we observe that realistic frames are generated but without smoothness in time.



Figure B.3.: Video frames unrolled in time of Fig. 5.20 for S-GAN: CNN-LSTM. When using S-GAN, *i.e.* a purely spatial ConvNet, we observe that realistic frames are generated but without smoothness in time.



Figure B.4.: Video frames unrolled in time of Fig. 5.20 for S-GAN: FCN-LSTM. When using S-GAN, *i.e.* a purely spatial ConvNet, we observe that realistic frames are generated but without smoothness in time.



Figure B.5.: Unrolled video frames of Fig. 5.20 for ST-GAN: CNN. The ST-GAN generates a realistic video.

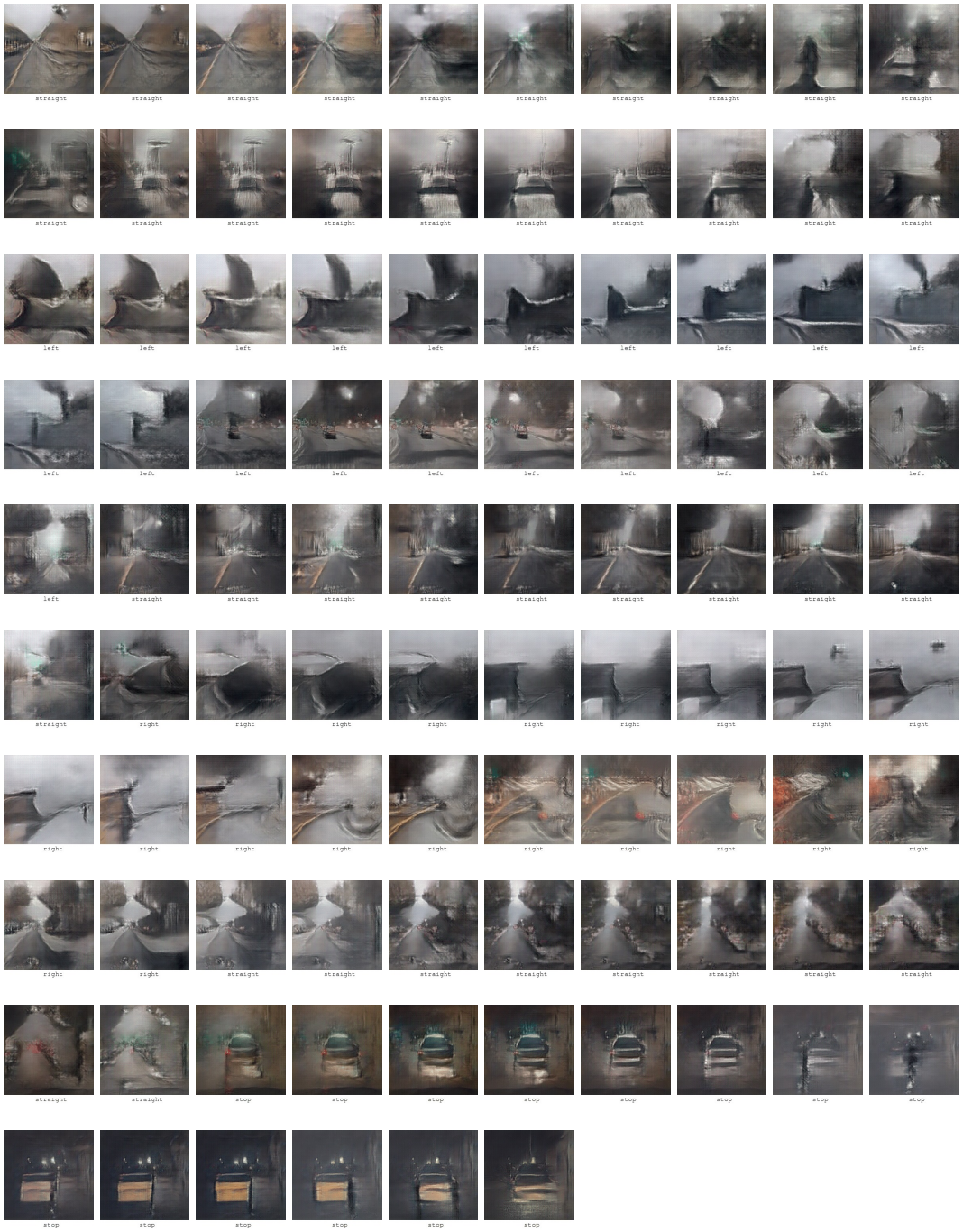


Figure B.6.: Video frames unrolled in time of Fig. 5.20 for ST-GAN: CNN-LSTM. The ST-GAN generates a realistic video.

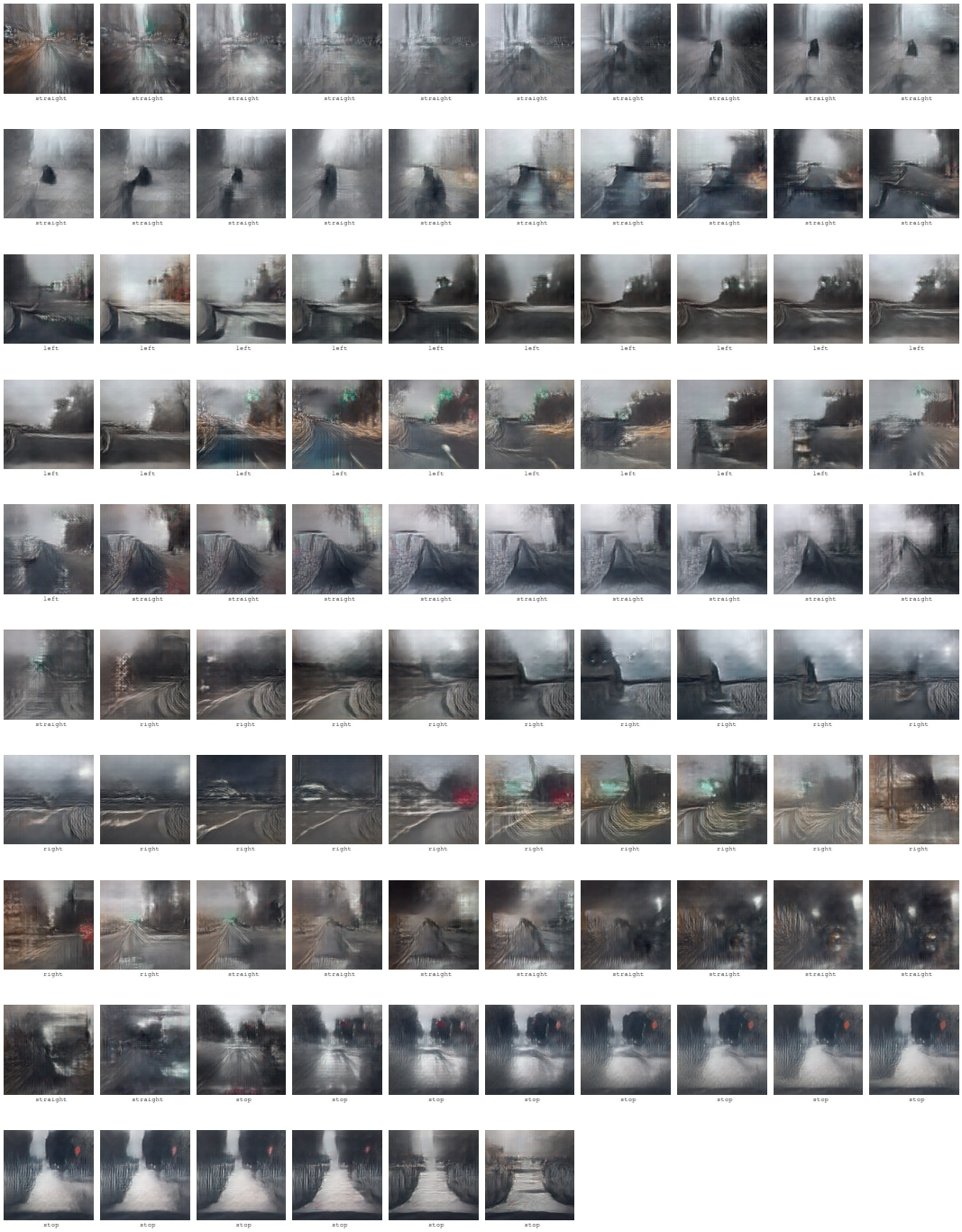


Figure B.7.: Video frames unrolled in time of Fig. 5.20 for ST-GAN: FCN-LSTM. The ST-GAN generates a realistic video.

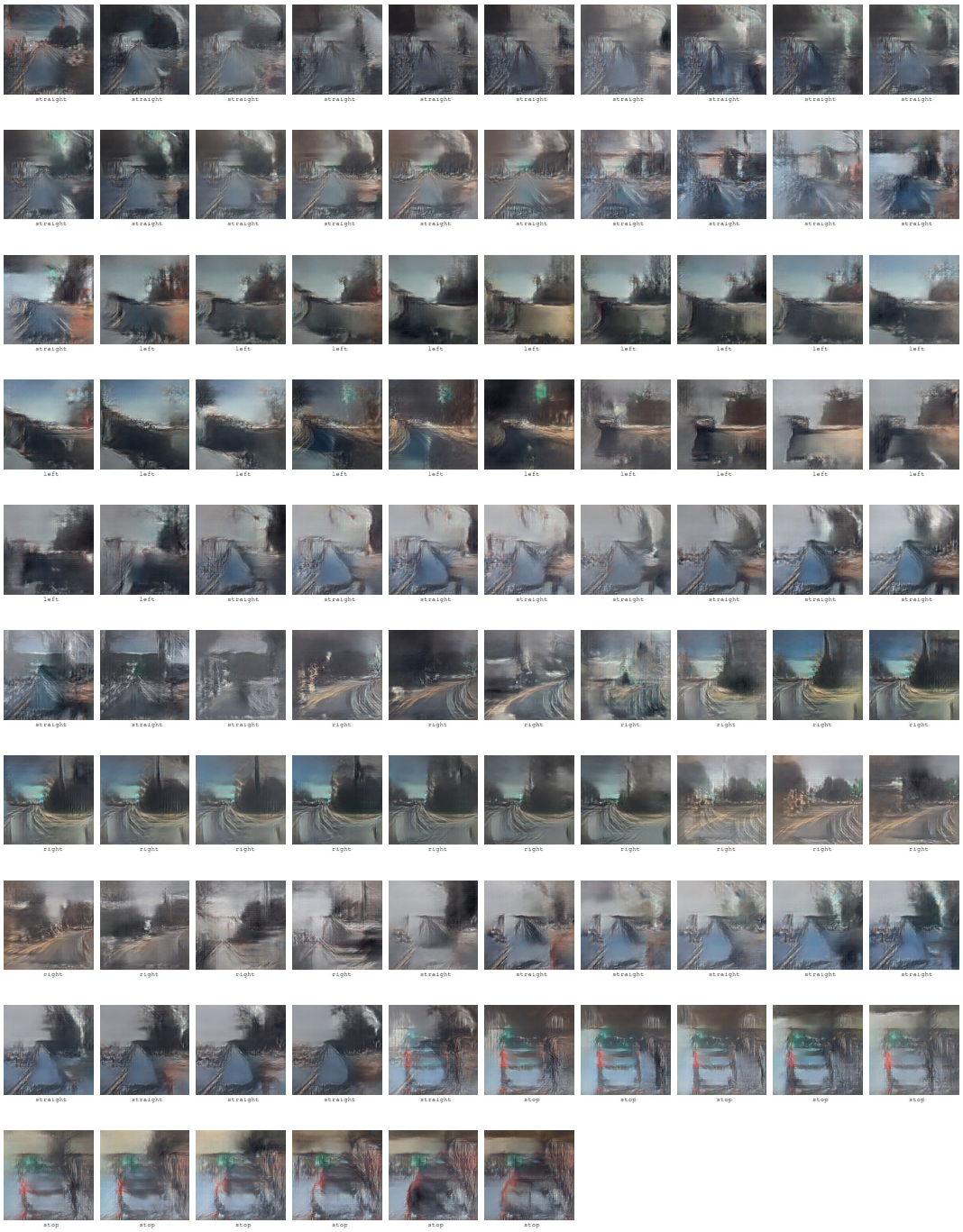


Figure B.8.: Video frames unrolled in time of the short-term approach showed in Fig. 5.21. We observe that concatenation of generated clips produces scene cuts with strong appearance changes between the individual clips. The video is generated with a sequence length of $T = 32$ frames.



Figure B.9.: Video frames unrolled in time of the temporal convolution approach Ψ_{tconv} for long-term generation showed in Fig. 5.21. This approach produces temporally coherent video samples with smooth transitions in time. The video is generated with a sequence length of $T = 16$ frames.



Figure B.10.: Video frames unrolled in time of the seq2seq approach Ψ_{seq2seq} for long-term generation showed in Fig. 5.21. This approach creates temporally coherent long-term videos with diversity in scenery. The video is generated with a sequence length of $T = 16$ frames.



Figure B.11.: Video frames unrolled in time of the long-term video generation showed in Fig. 5.21 using the future prediction module, $\Psi'_{seq2seq}$, which is only conditioned on the past. This approach creates temporally coherent long-term videos with diversity in scenery. The video is generated with a sequence length of $T = 16$ frames.



Figure B.12.: Video frames unrolled in time of the continuous BDD model [124] showed in Fig. 5.23. We see that red traffic lights and cars with flashing brake lights appear in multiple frames. That is, because we only maximize output neurons for steering directions without influence on the units that output vehicle speed.

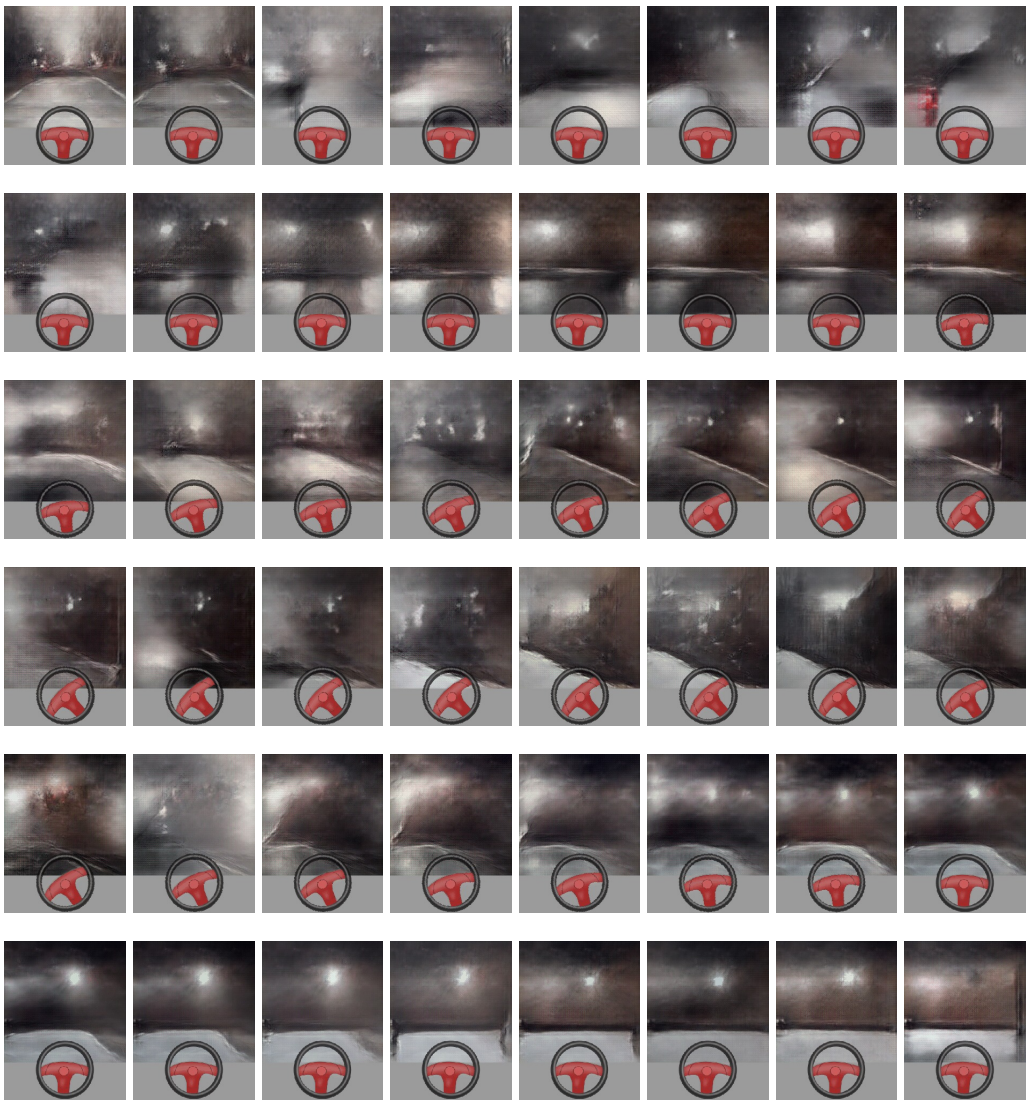


Figure B.13.: Video frames unrolled in time of the continuous ResNet-50 model [42] showed in Fig. 5.23. This model seems to only focus on near road structures *i.e.* on the bottom half of the visual field.

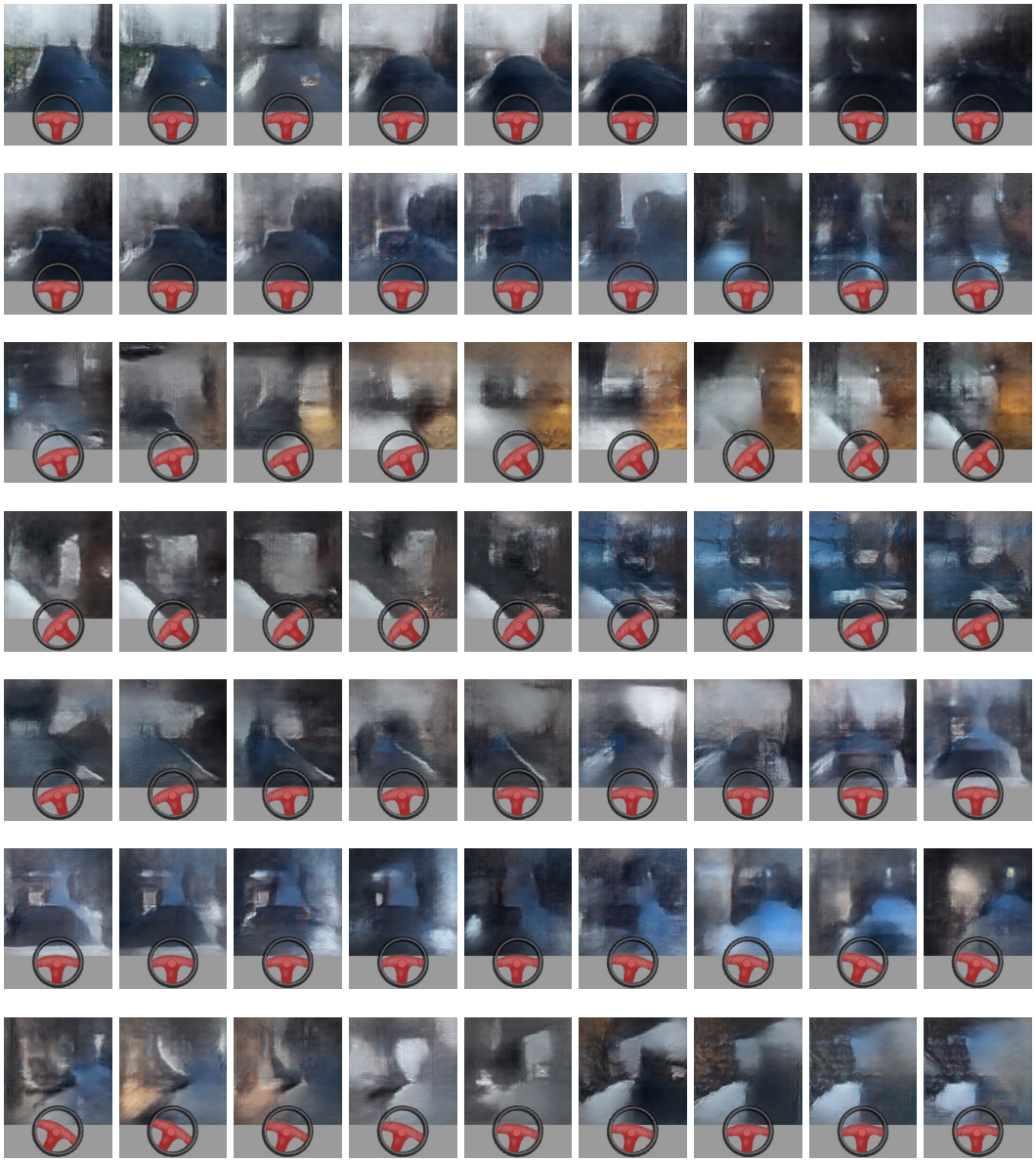
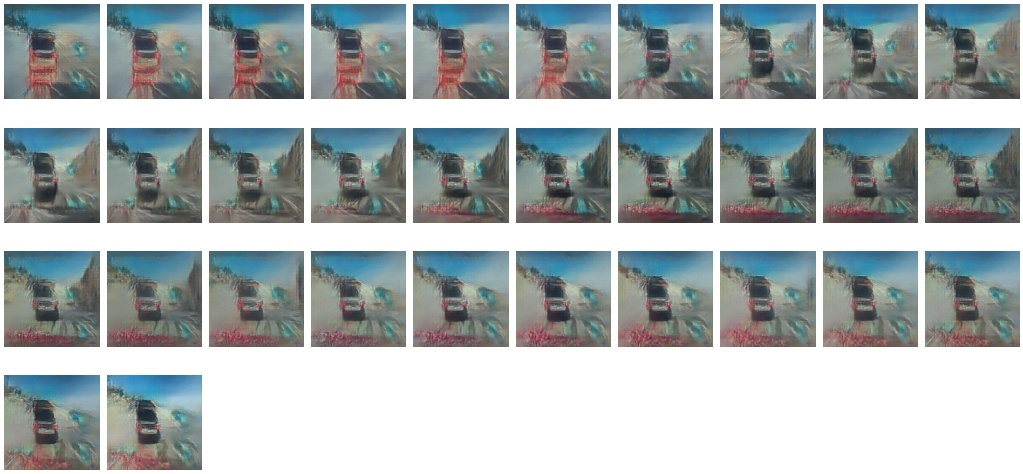
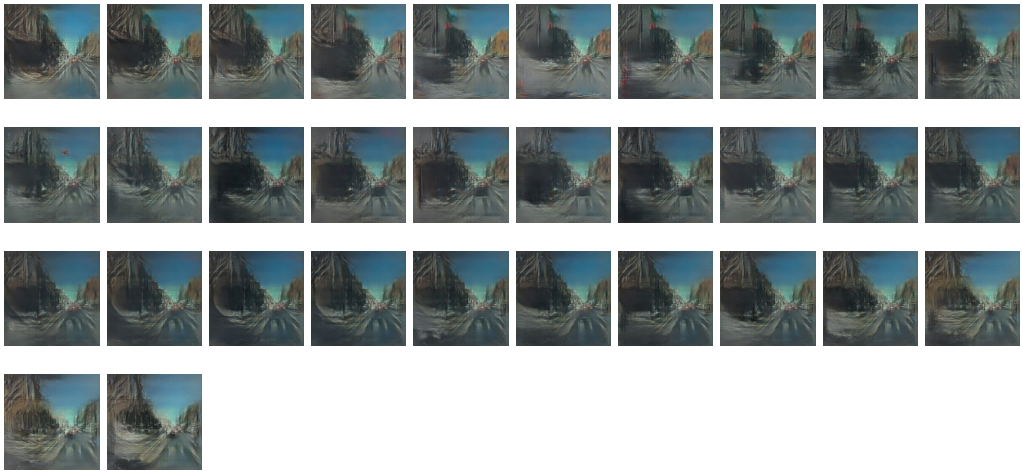


Figure B.14.: Video frames unrolled in time of the continuous Nvidia model [6] showed in Fig. 5.23. This model seems to only focus on near road structures *i.e.* on the bottom half of the visual field.

daylight: car



daylight: street



daylight: crossway

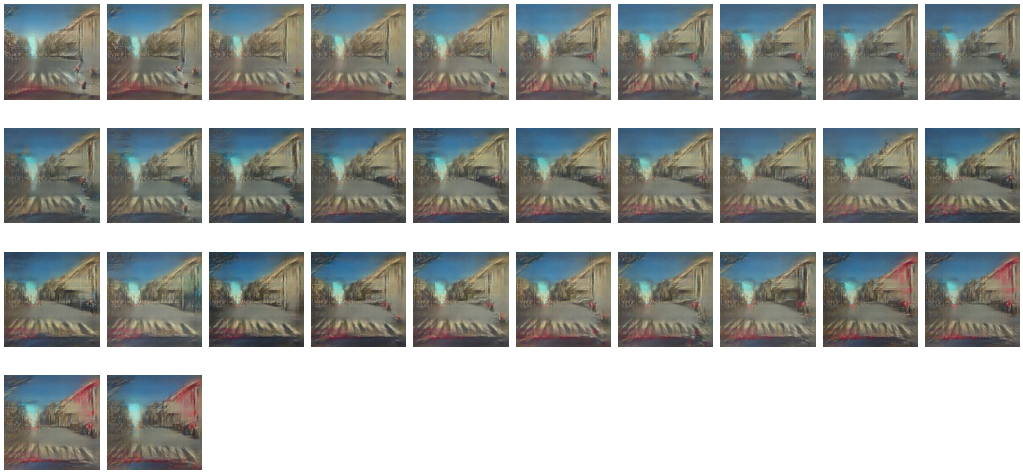
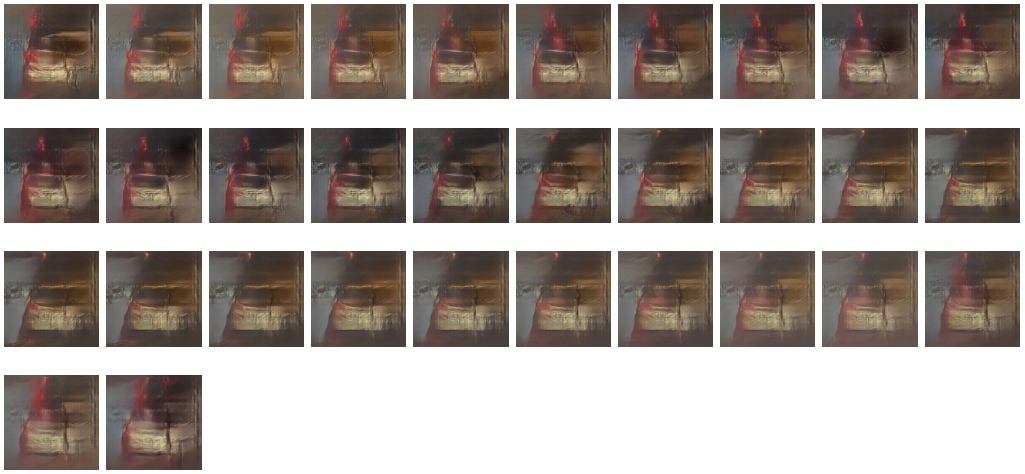
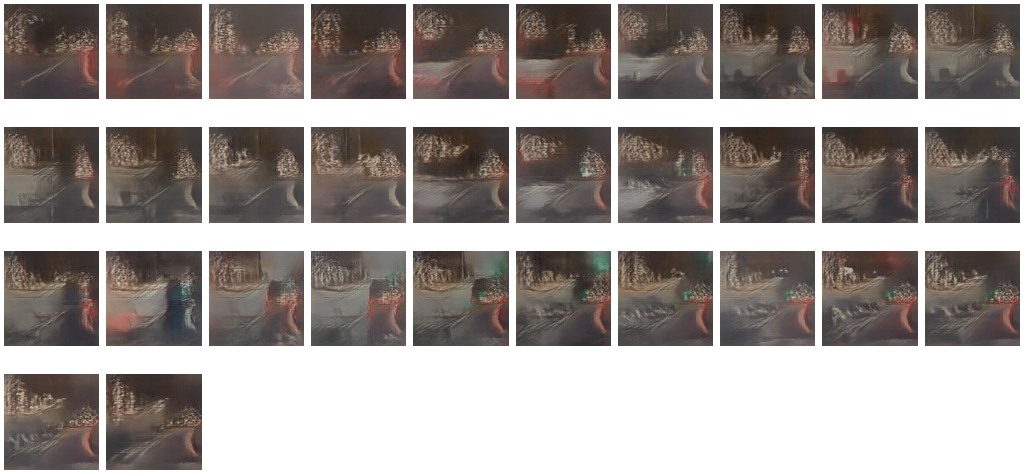


Figure B.15.: Video frames unrolled in time of Fig. 5.26 for hidden units in layer fc6 conditioned on the FCN-LSTM driving model [124]. In the daylight scenes blue sky attracts attention. For the 'car' scene we see the flashing brake light of the car in front. For 'daylight: street' we recognize cars in the distance and buildings on the side of the street. The sequence with the crosswalk in daylight shows how we wait at a red traffic light.

night: car



night: street

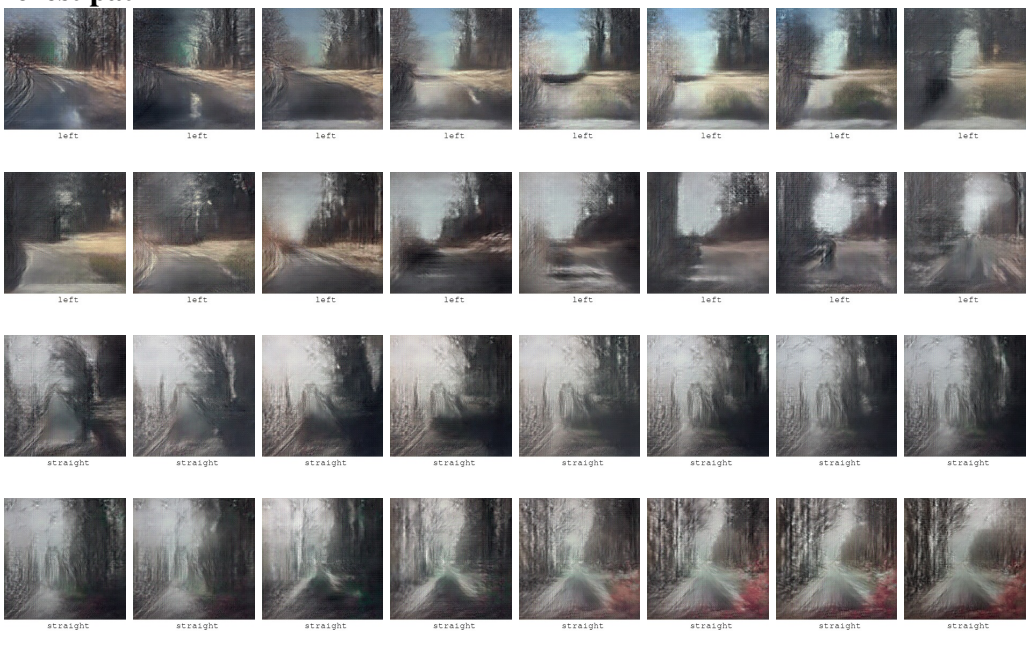


night: crossway



Figure B.16.: Video frames unrolled in time of Fig. 5.26 for hidden units in layer fc6 conditioned on the FCN-LSTM driving model [124]. In 'night: car' we see a stopped car close in front of our windshield and a red traffic light above the car. In the night scene of 'street' we notice road markings and a car appearing in the distance for a short moment. For 'night: crosswalk' we observe people crossing the street on the crosswalk while we also see green traffic lights.

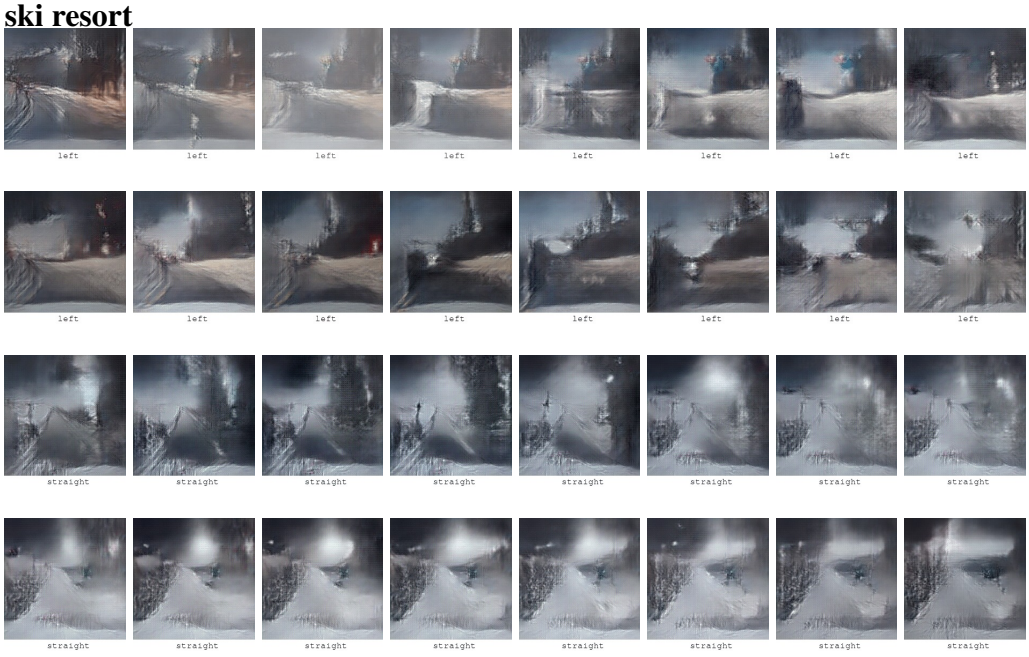
forest path





Dataset sample

ski resort

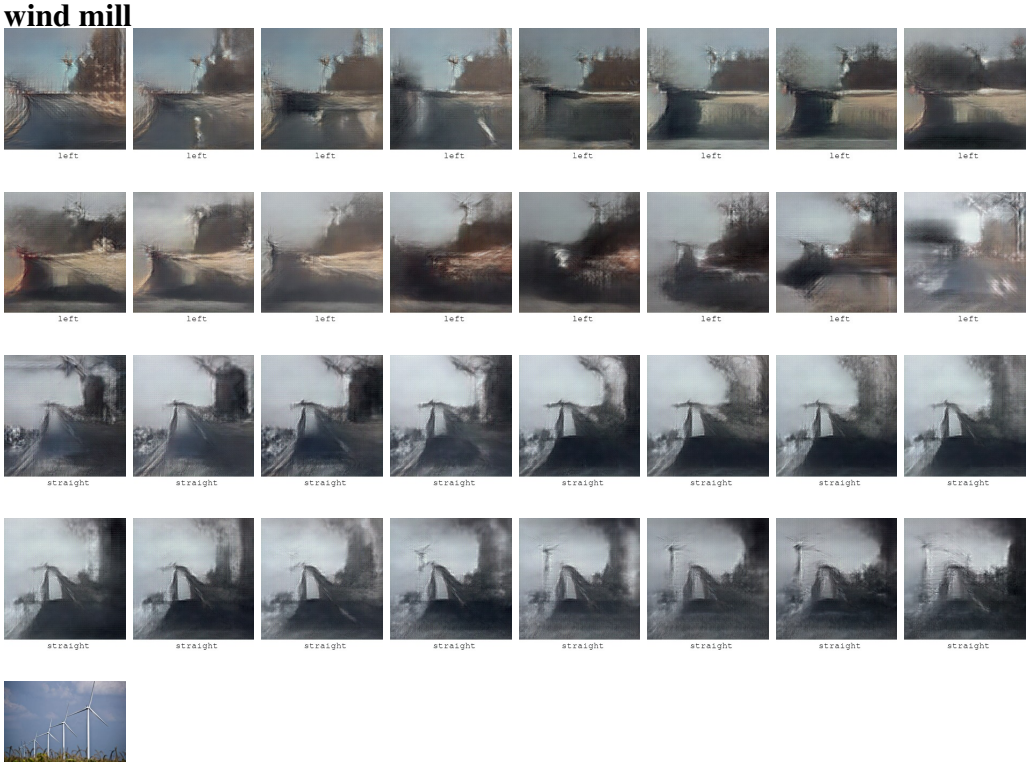




Dataset sample

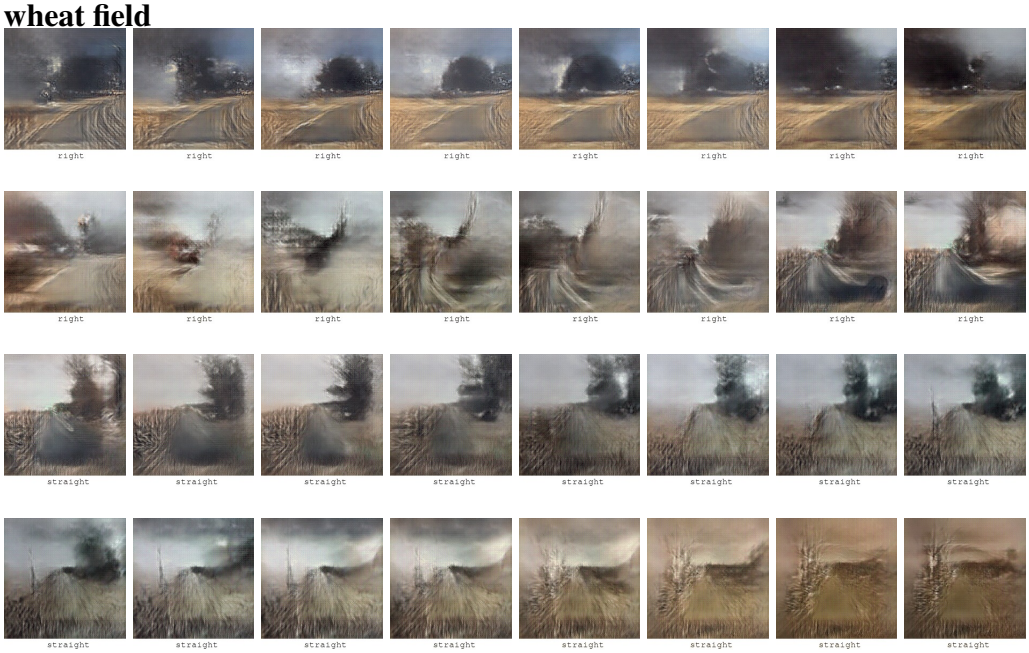
Figure B.17.: Video frames unrolled in time of Fig. 5.33 with driving sceneries for forest path and ski resort. The dataset sample shows an example how the training data of the scene model C [134] looks like for the respective class; this sample is not used in the video generation.

wind mill



Dataset sample

wheat field

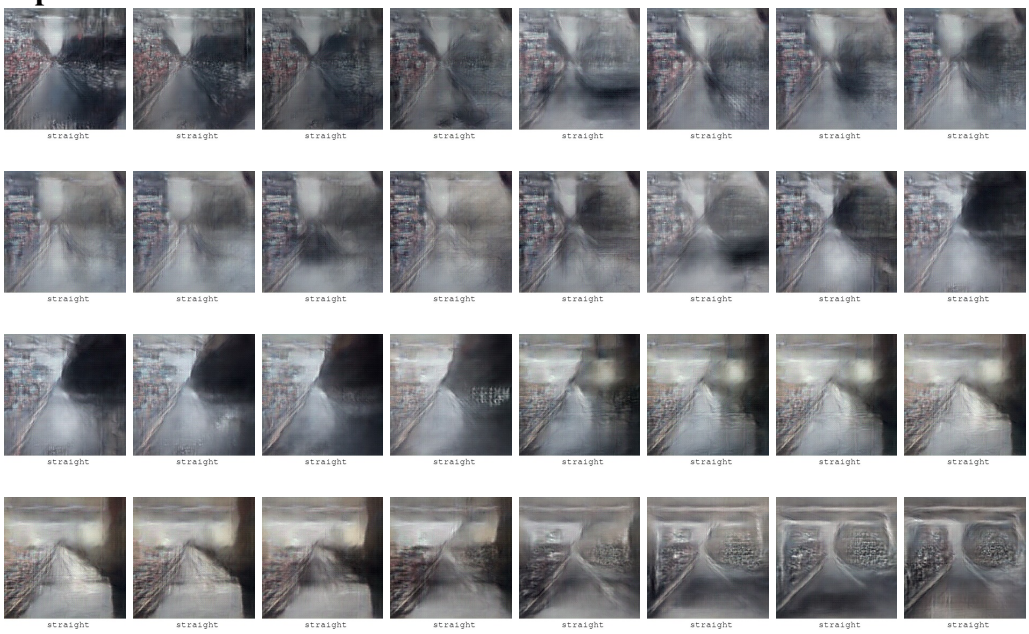




Dataset sample

Figure B.18.: Video frames unrolled in time of Fig. 5.33 with driving sceneries for wheat wind mill and wheat field. The dataset sample shows an example how the training data of the scene model C [134] looks like for the respective class; this sample is not used in the video generation.

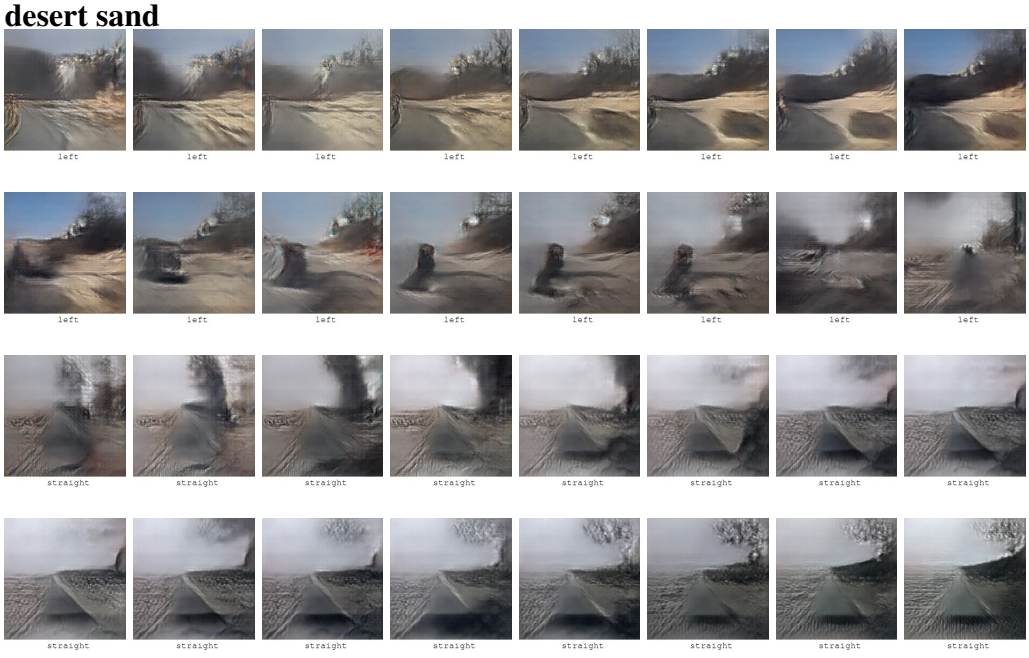
supermarket





Dataset sample

desert sand





Dataset sample

Figure B.19.: Video frames unrolled in time of Fig. 5.33 with driving sceneries for supermarket and desert sand. The dataset sample shows an example how the training data of the scene model C [134] looks like for the respective class; this sample is not used in the video generation.

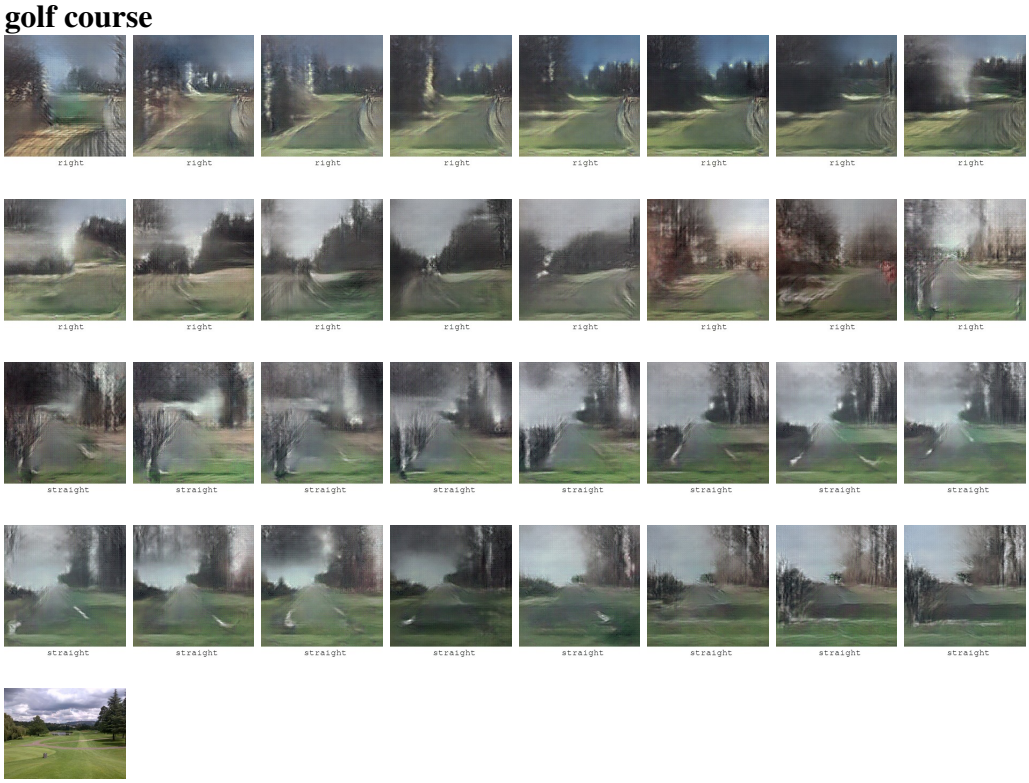
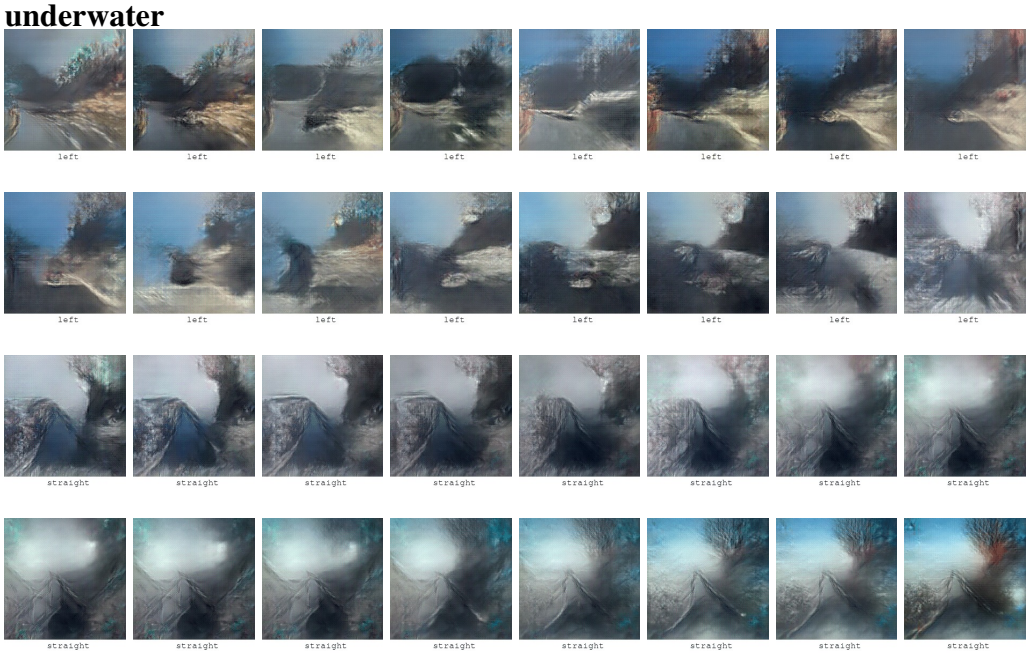


Figure B.20.: Video frames unrolled in time of Fig. 5.33 with driving sceneries for golf course and cathedral. The dataset sample shows an example how the training data of the scene model C [134] looks like for the respective class; this sample is not used in the video generation.

underwater





Dataset sample

Figure B.21.: Video frames unrolled in time of Fig. 5.33 with driving sceneries for underwater. The dataset sample shows an example how the training data of the scene model C [134] looks like for the respective class; this sample is not used in the video generation.

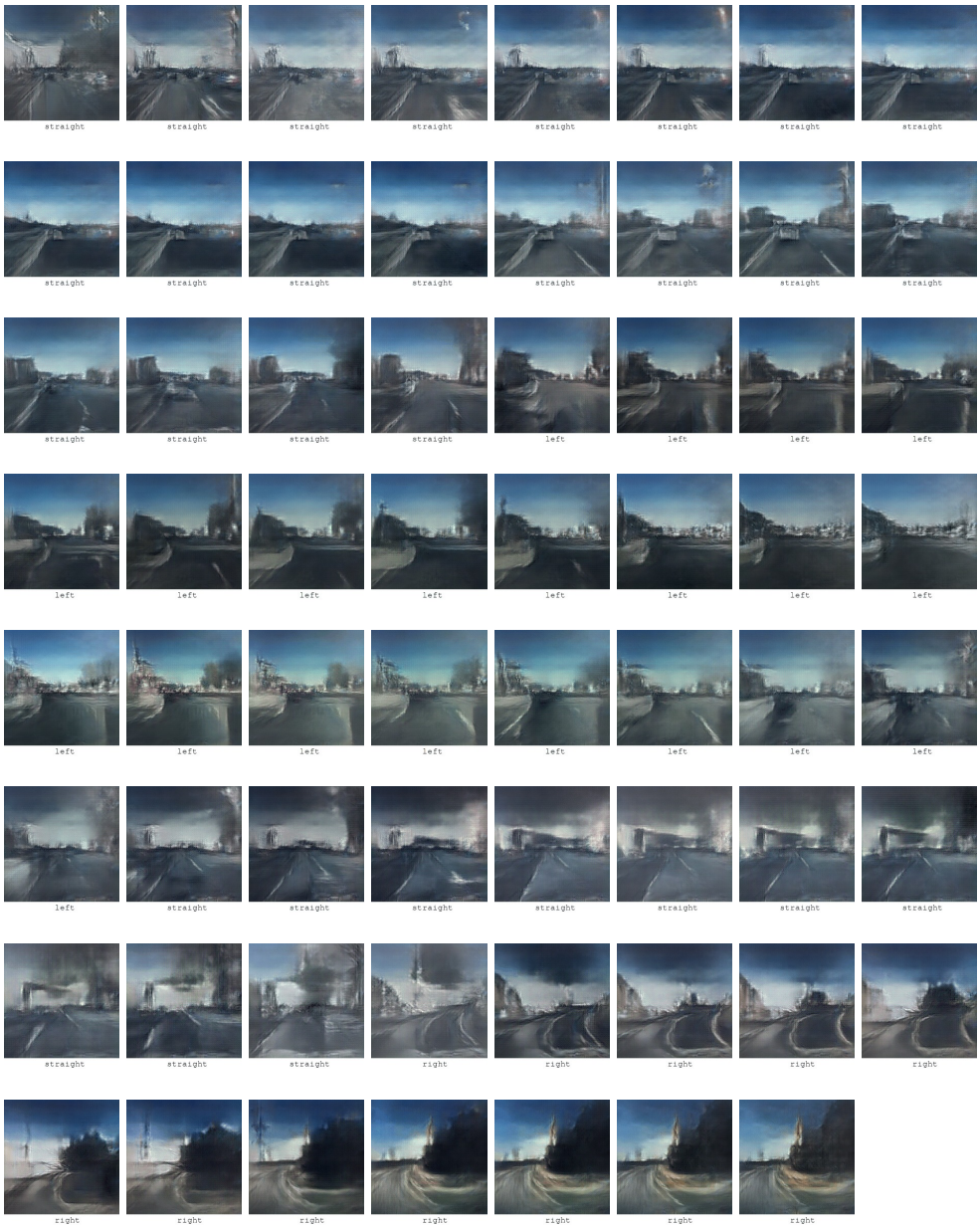


Figure B.22.: Video frames unrolled in time of a driving sequence initialized with a highway image as shown in Fig. 5.34.

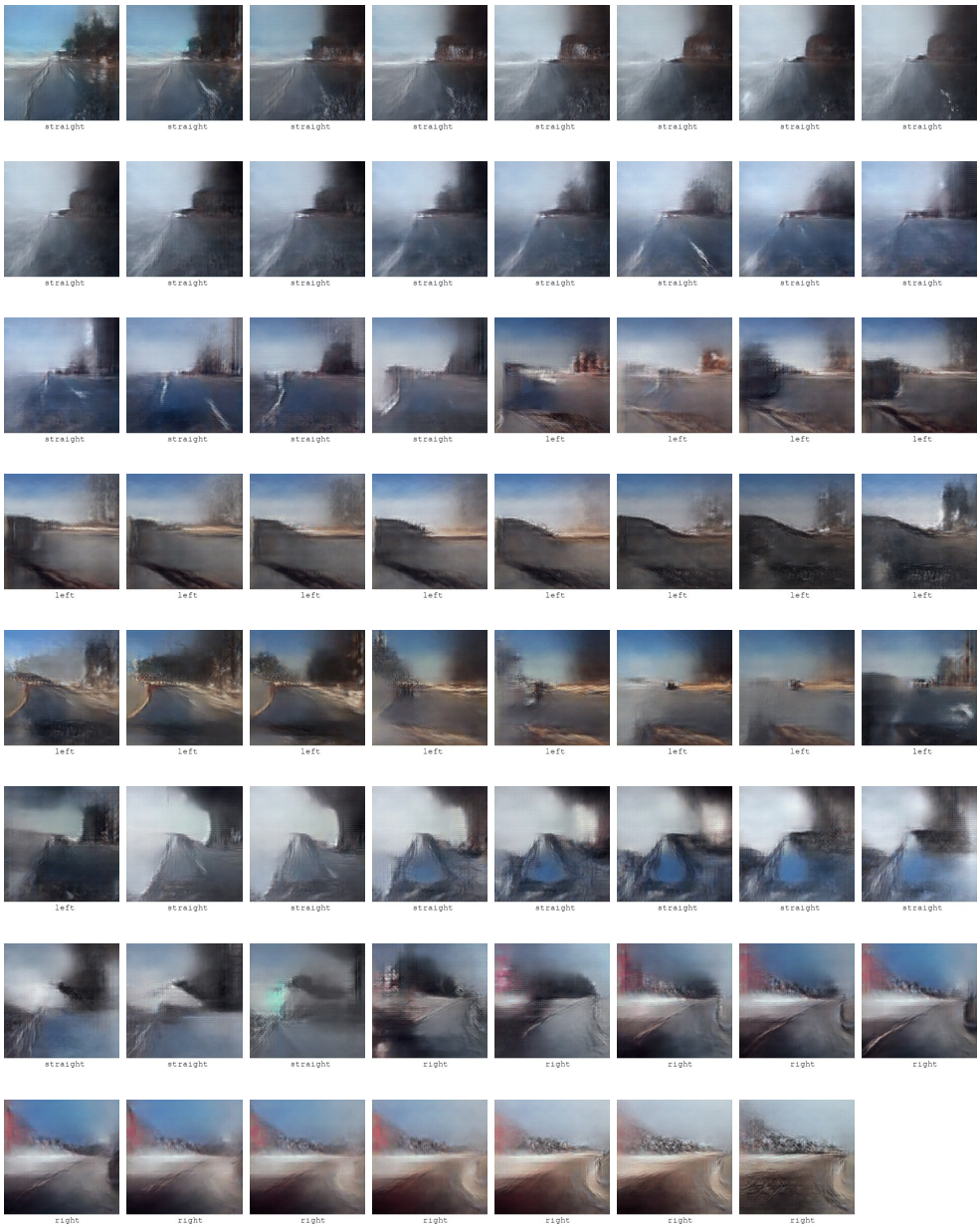


Figure B.23.: Video frames unrolled in time of a driving sequence initialized with a blue-gray drawing as shown in Fig. 5.34.

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv preprint arXiv:1511.00561*, 2015.

[3] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Proc. CVPR*, 2017.

[4] R. Biasini, G. Hotz, S. Khalandovsky, E. Santana, and N. van der Westhuizen. comma.ai self-driving car dataset. `https://github.com/commaai/research`, 2016. Accessed on: 2017/12/05.

[5] M. Bojarski, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, U. Muller, and K. Zieba. Visualbackprop: visualizing cnns for autonomous driving. *arXiv preprint arXiv:1611.05418*, 2016.

[6] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

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

[8] J. Bruna, P. Sprechmann, and Y. LeCun. Super-resolution with deep convolutional sufficient statistics. In *Proc. ICLR*, 2016.

[9] O. Cameron. Teaching a Machine to Steer a Car. https://medium.com/udacity/teaching-a-machine-to-steer-a-car-d73217f2492c, 2016. Accessed on: 2017/12/05.

[10] C. Chen, A. Seff, A. Kornhauser, and J. Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proc. CVPR*, 2015.

[11] Q. Chen and V. Koltun. Photographic image synthesis with cascaded refinement networks. *Proc. ICCV*, 2017.

[12] S. Chen, S. Zhang, J. Shang, B. Chen, and N. Zheng. Brain inspired cognitive model with attention for self-driving cars. *arXiv preprint arXiv:1702.05596*, 2017.

[13] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun. Monocular 3d object detection for autonomous driving. In *Proc. CVPR*, June 2016.

[14] Y. Chen, R. Ranftl, and T. Pock. A bi-level view of inpainting-based image compression. *arXiv preprint arXiv:1401.4112*, 2014.

[15] L. Chi and Y. Mu. Deep steering: Learning end-to-end driving model from spatial and temporal visual cues. *arXiv preprint arXiv:1708.03798*, 2017.

[16] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *EMNLP*, 2014.

[17] F. Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015. Accessed on: 2017/12/05.

[18] D. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *CoRR*, abs/1511.07289, 2015.

[19] F. Codevilla, M. Müller, A. Dosovitskiy, A. López, and V. Koltun. End-to-end driving via conditional imitation learning. *arXiv preprint arXiv:1710.02410*, 2017.

[20] A. Colovic, P. Knöbelreiter, A. Shekhovtsov, and T. Pock. End-to-end training of hybrid cnn-crf models for semantic segmentation using structured learning. *Computer Vision Winter Workshop*, 2017.

[21] B. Dai, D. Lin, R. Urtasun, and S. Fidler. Towards diverse and natural image descriptions via a conditional gan. *Proc. ICCV*, 2017.

[22] E. L. Denton, S. Chintala, R. Fergus, et al. Deep generative image models using an laplacian pyramid of adversarial networks. In *NIPS*, 2015.

[23] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proc. CVPR*, pages 2625–2634, 2015.

[24] G. Doretto, A. Chiuso, Y. N. Wu, and S. Soatto. Dynamic textures. *IJCV*, 51(2):91–109, 2003.

[25] A. Dosovitskiy and T. Brox. Generating images with perceptual similarity metrics based on deep networks. In *NIPS*, pages 658–666, 2016.

[26] P. Drews, G. Williams, B. Goldfain, E. A. Theodorou, and J. M. Rehg. Aggressive deep driving: Combining convolutional neural networks and model predictive control. In *Conference on Robot Learning*, pages 133–142, 2017.

[27] S. Du, H. Guo, and A. Simpson. Self-driving car steering angle prediction based on image recognition. `cs231n.stanford.edu/reports/2017/pdfs/626.pdf`, 2017. Accessed on: 2017/12/05.

[28] H. M. Eraqi, M. N. Moustafa, and J. Honer. End-to-end deep learning for steering autonomous vehicles considering temporal dependencies. *arXiv preprint arXiv:1710.03804*, 2017.

[29] D. Erhan, Y. Bengio, A. Courville, and P. Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341:3, 2009.

[30] D. Erhan, A. Courville, and Y. Bengio. Understanding representations learned in deep architectures. *Department d'Informatique et Recherche Operationnelle, University of Montreal, QC, Canada, Tech. Rep*, 1355, 2010.

[31] C. Feichtenhofer, A. Pinz, and R. Wildes. Spatiotemporal residual networks for video action recognition. In *NIPS*, 2016.

[32] T. Fernando, S. Denman, S. Sridharan, and C. Fookes. Going deeper: Autonomous steering with neural memory networks. In *Proc. CVPR*, pages 214–221, 2017.

[33] R. Gadde, V. Jampani, and P. V. Gehler. Semantic video cnns through representation warping. In *Proc. ICCV*. IEEE, 2017.

[34] A. Ganesh, J. Charalel, M. D. Sarma, and N. Xu. Deep reinforcement learning for simulated autonomous driving. http://cs229.stanford.edu/proj2016/report/Ganesh-Charalel-DasSarma-Xu-DeepReinforcementLearningForSimulatedAutonomousDriving-report.pdf, 2016. Accessed on: 2017/12/05.

[35] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *Proc. CVPR*, 2016.

[36] A. Giusti, J. Guzzi, D. C. Cireșan, F.-L. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro, et al. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters*, 1(2):661–667, 2016.

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

[38] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *NIPS*, 2014.

[39] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra. Draw: A recurrent neural network for image generationf. *arXiv preprint arXiv:1502.04623*, 2015.

[40] S. Guadarrama and N. Silberman. Tensorflow-slim. https://github.com/tensorflow/tensorflow/tree/master/tensorflow/contrib/slim, 2016. Accessed on: 2017/12/05.

[41] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik. Cognitive mapping and planning for visual navigation. *Proc. CVPR*, 2017.

[42] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. CVPR*, 2016.

[43] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[44] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. In *Spin Glass Theory and Beyond: An Introduction to the Replica Method and Its Applications*, pages 411–415. World Scientific, 1987.

[45] Y. Hou, S. Hornauer, and K. Zipser. Fast recurrent fully convolutional networks for direct perception in autonomous driving. *arXiv preprint arXiv:1711.06459*, 2017.

[46] C. Innocenti, H. Lindén, G. Panahandeh, L. Svensson, and N. Mohammadiha. Imitation learning for vision-based lane keeping assistance. *arXiv preprint arXiv:1709.03853*, 2017.

[47] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. *arXiv preprint arXiv:1611.07004*, 2016.

[48] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proc. ACMM*, 2014.

[49] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *Proc. ECCV*, 2016.

[50] N. Kalchbrenner, A. v. d. Oord, K. Simonyan, I. Danihelka, O. Vinyals, A. Graves, and K. Kavukcuoglu. Video pixel networks. *arXiv preprint arXiv:1610.00527*, 2016.

[51] T. Karras, T. Aila, S. Laine, and J. Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[52] J. Kim and J. Canny. Interpretable learning for self-driving cars by visualizing causal attention. *arXiv preprint arXiv:1703.10631*, 2017.

[53] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *Proc. ICLR*, 2015.

[54] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *Proc. ICLR*, 2014.

[55] R. Kotikalapudi and contributors. keras-vis. `https://github.com/raghakot/keras-vis`, 2017. Accessed on: 2017/12/05.

[56] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012.

[57] A. Kundu, Y. Li, F. Dellaert, F. Li, and J. M. Rehg. Joint semantic segmentation and 3d reconstruction from monocular video. In D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors, *Proc. ECCV*, 2014.

[58] H. Larochelle and I. Murray. The neural autoregressive distribution estimator. In *Proc. AISTATS*, pages 29–37, 2011.

[59] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[60] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proc. CVPR*, 2017.

[61] N. Lee, W. Choi, P. Vernaza, C. B. Choy, P. H. S. Torr, and M. K. Chandraker. DESIRE: distant future prediction in dynamic scenes with interacting agents. *arXiv preprint arXiv:1704.04394*, 2017.

[62] C. Li and M. Wand. Precomputed real-time texture synthesis with markovian generative adversarial networks. In *Proc. ECCV*. Springer, 2016.

[63] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *Proc. ECCV*, 2014.

[64] Z. Liu and Y. Wang. Deeper direct perception in autonomous driving. `http://cs231n.stanford.edu/reports/2016/pdfs/123_Report.pdf`, 2016. Accessed on: 2017/12/05.

[65] W. Lotter, G. Kreiman, and D. Cox. Deep predictive coding networks for video prediction and unsupervised learning. *arXiv preprint arXiv:1605.08104*, 2016.

[66] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *JMLR*, 9(Nov):2579–2605, 2008.

[67] A. Mahendran and A. Vedaldi. Visualizing deep convolutional neural networks using natural pre-images. *IJCV*, 120(3):233–255, 2016.

[68] M. Malinowski, M. Rohrbach, and M. Fritz. Ask your neurons: A neural-based approach to answering questions about images. In *Proc. ICCV*, December 2015.

[69] M. Mathieu, C. Couprie, and Y. LeCun. Deep multi-scale video prediction beyond mean square error. *arXiv preprint arXiv:1511.05440*, 2015.

[70] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[71] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[72] A. Mordvintsev, C. Olah, and M. Tyka. Inceptionism: Going deeper into neural networks. *Google Research Blog. Retrieved June*, 20:14, 2015.

[73] A. Y. Ng and M. I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *NIPS*, pages 841–848, 2002.

[74] A. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, and J. Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. In *NIPS*, 2016.

[75] A. Nguyen, J. Yosinski, Y. Bengio, A. Dosovitskiy, and J. Clune. Plug & play generative networks: Conditional iterative generation of images in latent space. *Proc. CVPR*, 2017.

[76] A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proc. CVPR*, 2015.

[77] A. Nguyen, J. Yosinski, and J. Clune. Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks. *arXiv preprint arXiv:1602.03616*, 2016.

[78] A. Odena, C. Olah, and J. Shlens. Conditional image synthesis with auxiliary classifier gans. *arXiv preprint arXiv:1610.09585*, 2016.

[79] J. Oh, V. Chockalingam, S. Singh, and H. Lee. Control of memory, active perception, and action in minecraft. *arXiv preprint arXiv:1605.09128*, 2016.

[80] J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh. Action-conditional video prediction using deep networks in atari games. In *NIPS*, 2015.

[81] C. Olah, A. Mordvintsev, and L. Schubert. Feature visualization. *Distill*, 2017. https://distill.pub/2017/feature-visualization.

[82] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. *Proc. ICML*, 2016.

[83] A. v. d. Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu. Conditional image generation with pixelcnn decoders. *NIPS*, 2016.

[84] D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In D. S. Touretzky, editor, *NIPS*, pages 305–313. Morgan-Kaufmann, 1989.

[85] T. Pärnamaa and Team Rambo. Udacity challenge 2 - rambo. https://github.com/udacity/self-driving-car/tree/master/steering-models/community-models/rambo, 2016. Accessed on: 2017/12/05.

[86] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, 2009.

[87] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *Proc. ICLR*, 2015.

[88] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. Generative adversarial text to image synthesis. *Proc. ICML*, 2016.

[89] M. Ruder, A. Dosovitskiy, and T. Brox. Artistic style transfer for videos and spherical images. *arXiv preprint arXiv:1708.04538*, 2017.

[90] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 115(3):211–252, 2015.

[91] SAE INTERNATIONAL. Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems (j3016). `https://www.sae.org/misc/pdfs/automated_driving.pdf`, 2014. Accessed on: 2017/12/05.

[92] M. Saito, E. Matsumoto, and S. Saito. Temporal generative adversarial nets with singular value clipping. In *Proc. ICCV*, 2017.

[93] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. nips, 2016. *Proc. CVPR*, 2017.

[94] E. Santana and G. Hotz. Learning a driving simulator. *arXiv preprint arXiv:1608.01230*, 2016.

[95] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *arXiv preprint arXiv:1610.02391*, 2016.

[96] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[97] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[98] K. Soomro, A. R. Zamir, and M. Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[99] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. In *Proc. ICLR*, 2015.

[100] N. Srivastava, E. Mansimov, and R. Salakhudinov. Unsupervised learning of video representations using lstms. In *Proc. ICML*, 2015.

[101] S. Sukhbaatar, J. Weston, R. Fergus, et al. End-to-end memory networks. In *NIPS*, 2015.

[102] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

[103] C. Szegedy, A. Toshev, and D. Erhan. Deep neural networks for object detection. In *NIPS*, 2013.

[104] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proc. CVPR*, pages 2818–2826, 2016.

[105] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[106] M. Szummer and R. W. Picard. Temporal texture modeling. In *Intl. Conf. Image Proc.* IEEE, 1996.

[107] M. Teichmann, M. Weber, M. Zoellner, R. Cipolla, and R. Urtasun. Multinet: Real-time joint semantic reasoning for autonomous driving. *arXiv preprint arXiv:1612.07695*, 2016.

[108] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[109] S. Tulyakov, M.-Y. Liu, X. Yang, and J. Kautz. Mocogan: Decomposing motion and content for video generation. *arXiv preprint arXiv:1707.04993*, 2017.

[110] Udacity. Public driving dataset. `https://www.udacity.com/self-driving-car`, 2016. Accessed on: 2017/12/05.

[111] L. Van Der Maaten. Accelerating t-sne using tree-based algorithms. *JMLR*, 15(1):3221–3245, 2014.

[112] S. Venugopalan, M. Rohrbach, J. Donahue, R. Mooney, T. Darrell, and K. Saenko. Sequence to sequence-video to text. In *Proc. ICCV*, 2015.

[113] R. Villegas, J. Yang, S. Hong, X. Lin, and H. Lee. Decomposing motion and content for natural video sequence prediction. In *Proc. ICLR*, 2017.

[114] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. In *Proc. CVPR*, pages 3156–3164, 2015.

[115] C. Vondrick, H. Pirsiavash, and A. Torralba. Generating videos with scene dynamics. In *NIPS*, 2016.

[116] C. Vondrick and A. Torralba. Generating the future with adversarial transformers. In *Proc. CVPR*, 2017.

[117] X. Wang and A. Gupta. Generative image modeling using style and structure adversarial networks. In *Proc. ECCV*, 2016.

[118] D. Wei, B. Zhou, A. Torrabla, and W. Freeman. Understanding intra-class knowledge inside cnn. *arXiv preprint arXiv:1507.02379*, 2015.

[119] G. Wei, D. Hus, W. S. Lee, S. Shen, and K. Subramanian. Intention-net: Integrating planning and deep learning for goal-directed autonomous navigation. *arXiv preprint arXiv:1710.05627*, 2017.

[120] J. Weston, S. Chopra, and A. Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

[121] S. Wold, K. Esbensen, and P. Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[122] L. Wolf, Y. Taigman, and A. Polyak. Unsupervised creation of parameterized avatars. *arXiv preprint arXiv:1704.05693*, 2017.

[123] W. Xiong, W. Luo, L. Ma, W. Liu, and J. Luo. Learning to generate time-lapse videos using multi-stage dynamic generative adversarial networks. *arXiv preprint arXiv:1709.07592*, 2017.

[124] H. Xu, Y. Gao, F. Yu, and T. Darrell. End-to-end learning of driving models from large-scale video datasets. *Proc. CVPR*, 2017.

[125] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proc. ICML*, 2015.

[126] T. Xue, J. Wu, K. Bouman, and B. Freeman. Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks. In *NIPS*, 2016.

[127] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. Understanding neural networks through deep visualization. *Proc. ICML*, 2015.

[128] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Proc. ECCV*, 2014.

[129] H. Zhang, T. Xu, H. Li, S. Zhang, X. Huang, X. Wang, and D. Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. *Proc. ICCV*, 2017.

[130] J. Zhang, Z. Lin, J. Brandt, X. Shen, and S. Sclaroff. Top-down neural attention by excitation backprop. In *Proc. ECCV*, 2016.

[131] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Object detectors emerge in deep scene cnns. In *Proc. ICLR*, 2014.

[132] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Learning deep features for discriminative localization. In *Proc. CVPR*, 2016.

[133] B. Zhou, A. Khosla, A. Lapedriza, A. Torralba, and A. Oliva. Places: An image database for deep scene understanding. *arXiv preprint arXiv:1610.02055*, 2016.

[134] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Learning deep features for scene recognition using places database. In *NIPS*, 2014.

[135] Y. Zhou and T. L. Berg. Learning temporal transformations from time-lapse videos. In *Proc. ECCV*, 2016.

[136] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *Proc. ICCV*, 2017.