Mercedes-Benz
Research & Development North America, Inc.

Deep Prediction Learning for Autonomous Driving

T2000 Report

for the

Bachelor of Engineering

from the Course of Studies IT-Automotive

at the Cooperative State University Baden-Württemberg Stuttgart

by

Phillip Lippe

18. August 2017

Time of Project	05/29/2017 to 08/20/2017
Student ID, Course	9045534, ITA2015
Company	Daimler AG, Sunnyvale, USA
Supervisor in the Company	Yan Meng

Author's declaration

Hereby I solemnly declare:

1. that this T2000 Report, titled *Deep Prediction Learning for Autonomous Driving* is entirely the product of my own scholarly work, unless otherwise indicated in the text or references, or acknowledged below;
2. I have indicated the thoughts adopted directly or indirectly from other sources at the appropriate places within the document;
3. this T2000 Report has not been submitted either in whole or part, for a degree at this or any other university or institution;
4. I have not published this T2000 Report in the past;
5. the printed version is equivalent to the submitted electronic one.

I am aware that a dishonest declaration will entail legal consequences.

Sunnyvale, 18. August 2017

Phillip Lippe

Abstract

An autonomous driving vehicle has to consider the future trajectory of other traffic participants to plan its own safe and comfortable route. Therefore a prediction of all object's positions in the next few seconds is indispensable. However systems created "by hand" fail on the multitude of different traffic scenarios and driving behaviors. Considering the human as role model, the prediction is based on an enormous amount of samples that were seen before. This is why the following thesis deals with a machine learning approach for predicting trajectories of multiple objects in a traffic environment.

Artificial neural networks have outperformed on many different learning tasks, especially when dealing with images. Hence a new architecture, taking simple sensor data of the captured environment as input, is proposed in this thesis. Based on a recurrent encoder-decoder structure, the network analyzes the traffic scenario over multiple time steps and predict the next occupancy map containing all objects and possible free space. To generate longer sequences the output is fed back into the network while all predictions depend on the estimated trajectory for the ego vehicle. A combined loss function allows the network to track objects and consider even occluded vehicles for prediction.

In experiments the proposed network exceeds state-of-the-art models for video prediction and shows promising results. Lane following, even for turns, and overtaking of other traffic participants does the network predict quite well. Still, open challenges like the prediction of unseen objects and multimodal trajectories remain because of the task's complexity.

Contents

1 Introduction **1**
1.1 Motivation 1
1.2 Proposed Approach 3
1.3 Outline 5

2 Foundations **6**
2.1 Machine Learning 6
2.2 Artificial Neural Networks 10
2.3 Recurrent Neural Networks 15
2.4 Related work 18

3 Deep Prediction Learning **24**
3.1 Environment datasets 24
3.2 Learning predictions on moving ego vehicle 28

4 Experimental results **41**
4.1 MCNet for environment prediction 41
4.2 Prediction evaluation on moving vehicle 47
4.3 Comparison and Discussion 54

5 Outlook **57**

6 Conclusion **60**

A Results of MCNet **67**

B Results of prediction network **68**

List of Figures

1 Prediction of traffic participants [82] . . . 1
2 Prediction system concept . . . 4
3 Relations of T, P and E . . . 6
4 Supervised learning examples [66, 12] . . . 9
5 Reinforcement learning [76] . . . 10
6 Artificial neuron architecture [13] . . . 11
7 Comparison of activation functions . . . 12
8 Network structures [13] . . . 13
9 Convolution operation [38] . . . 14
10 Pooling layer [2] . . . 15
11 Unrolled RNN [57] . . . 16
12 Basic structure of LSTM [57] . . . 17
13 Idea of deepTracking [58] . . . 19
14 Minimax game of GANs . . . 21
15 Motion-Content Network [80] . . . 22
16 CDNA video prediction model [19] . . . 23
17 Environment representation [61] . . . 24
18 Bertha's sensor architecture [86] . . . 25
19 Sample gridmaps . . . 26
20 Generated occupancy/occlusion maps . . . 27
21 Motion maps . . . 28
22 Network architecture for moving ego vehicle predictions . . . 30
23 Structure of dilated dense block . . . 31
24 Sample keep-alive function . . . 33
25 Prediction framework . . . 35
26 Recurrent discriminator architecture . . . 37
27 Sharpening loss function . . . 38
28 Prediction framework . . . 39
29 PSNR error for static ego vehicle . . . 43
30 Occlusion problem for static sequences . . . 44
31 MCNet predictions for moving vehicle . . . 46
32 Result without action-conditional prediction . . . 49
33 Action-conditional sequence with lower frame rate . . . 50
34 Traffic scenario overfitting . . . 51
35 Comparison using sharpening loss or not . . . 53
36 Object representation used by DESIRE [45] . . . 57
37 Pixel loss evaluation . . . 58
38 XNOR convolution approximation [62] . . . 59
39 MCNet predictions for moving vehicle . . . 67
40 Recognizing lane assignment . . . 68
41 Multimodal distributions in predictions . . . 68

List of Equations

2.1 Mean-squared error [24] . . . 8
2.2 Joint distribution in unsupervised learning [24] . . . 9
2.3 Convert supervised conditional distribution to joint distribution [24] . . . 9
2.4 Mathematical artificial neuron [13] . . . 11
2.5 LSTM forget gate [30] . . . 17
2.6 LSTM input gate [30] . . . 17
2.7 LSTM output gate [30] . . . 18
2.8 GAN value function [25] . . . 21
2.9 Wasserstein GAN value function [27] . . . 21
3.2 Transformation matrix for moving vehicle [22] . . . 34
3.3 Cross-entropy loss function [55] . . . 36
3.4 Weighted binary cross-entropy for network training . . . 36
3.5 Linear sharpening loss function . . . 38
3.6 Sharpening loss function . . . 38
4.1 Peak signal-to-noise ratio [50, 33] . . . 43

List of abbreviations

CDNA	Convolutional Dynamic Neural Advection [19]
CNN	Convolutional Neural Network [43]
ConvLSTM	Convolutional Long short-term memory [70]
GAN	Generative adversarial Network [25]
LSTM	Long short-term memory [30]
MCNet	Motion-Content Network [50]
MSE	Mean-squared error [24]
PSNR	Peak signal-to-noise ratio [80]
ReLU	Rectified Linear Units [13]
RNN	Recurrent Neural Network [24]
SELU	Scaled Exponential Linear Units [39]

List of Tables

1 Overview of MCNet training parameters for static ego vehicle 42
2 Overview of MCNet training parameters for moving ego vehicle 45
3 Overview of training parameters for proposed prediction network 48

1. Introduction

1.1. Motivation

The vision of autonomous driving is becoming step by step reality. Since 1987 a research project called PROMETHEUS [83] has successfully led to the first autonomous vehicle, many companies started to develop this technology further and make it ready for series production. So for example Mercedes Benz which presented 2013 the prototype "S500 Intelligent Drive". This system was able to drive the route between Mannheim and Pforzheim of about 100km autonomously equipped by only sensors out of series production [86].

However autonomous driving remains complex and difficult because there are so many possible scenarios that a car has to solve. Basically such a system has to encounter three main tasks: detection of its surrounding, understanding of the environment and therefore choose its action and trajectory. Multiple different sensors like camera, radar and lidar observe all objects around the ego vehicle. Because the output of these sensors only consist of pixels or point clouds, an interpretation system runs over the raw data to extract objects and other important information. As a last step the car has to plan a safe and comfortable trajectory where it will drive. This includes an estimation of the possible free space in a certain time period that is influenced by the other traffic participants. Therefore a prediction of trajectories for every individual obstacle in the environment, like figure 1 shows, is indispensable.

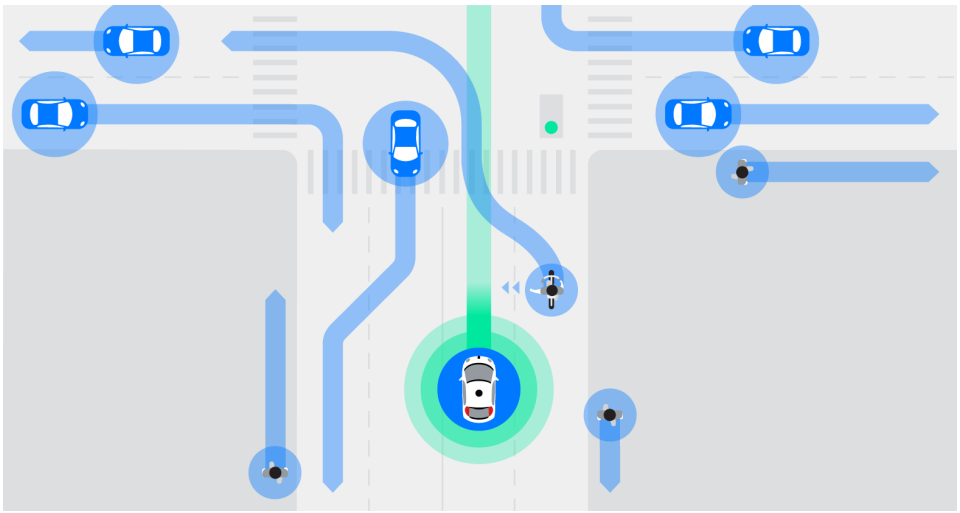


Figure 1: A typical urban traffic environment includes car, pedestrians and cyclists. All of them can potentially cross the planned trajectory of the ego vehicle and therefore have to be taken into account. In this example the hand signal of the right cyclist infers his intention to move to the left side of the lane. So the ego vehicle should slow down to let the cyclist pass safely and comfortably [82].

A human driver also predicts the behavior of its surrounding objects in order to handle potential dangerous situations. This is why a driver slowly passes pedestrians that are standing on the road and not paying attention to the traffic, or the merging process on highways rely on human intuition and interaction [60]. Especially intersections in crowded urban scene represent a challenge. Traffic lights, pedestrians, stop lines, traffic rules and much more have to be taken into account to predict the interactions between traffic participants and conclude the possible trajectories for the ego vehicle. In addition the car drive becomes more comfortable and economic based on the anticipated future.

Considering possible driving behaviors to the multitude of all different traffic scenarios it seems to be very hard, or even impossible, to create a system "by hand" that tackles every driving situation in a correct way [60]. Classical approaches like [49, 68] are mostly restricted to some constrains like specific behavior options so that it is not fully capable for real-world application. But how is a human then able to do that when computers with enormous computational power fail?

The answer is: by learning. A human driver has seen millions of different scenarios and also the corresponding followup actions of all traffic participants. Based on this experience they constantly adjust their prediction to perform even better next time. So when humans fail on one situation they learn from it until the predictions are improving to a sufficient state. Inspired by this behavior many big companies like Google [82], start ups like drive.ai [17] or labs like Berkeley DeepDrive [14] start to apply deep learning systems for autonomous driving. Since 2012, when the artificial neural network called AlexNet [40] outperformed the classification of images in one of 1,000 possible categories, machine learning became to a popular research topic and application for intelligent systems. First neural networks were used for image understanding in the context of self-driving cars to extract objects out of a RGB image [12]. Furthermore approaches were tested to imitate a human driver by only using one neural network to calculate out of either a front-view RGB image or an object representation of all traffic participants, the steering angle and acceleration [9, 41]. Nowadays machine learning applies for multiple tasks like language understanding [74, 75] or motion planning [20].

Machine learning rests on extracting features and patterns out of the input [24]. In computer vision every object has a specific pattern by which a human recognizes it. Mostly this feature is very complex and hard to develop, so that just learning it is the easiest way. Going back to the prediction task, all traffic environments also contain features that have be recognized. For example every driver has a certain driving style, but this depends on various factors. Some drivers are more skilled than others, some drive aggressive or some are even drunk. The prediction system has to adapt to all of them to estimate their future trajectory correctly. Common driver profiles could be implemented by hand but there are too many possible styles to capture all of them. A learning system does a much better job by extracting necessary features and clustering similar ones together [42]. In addition the road structure and interactions between obstacles can be seen as features. The combination of all patterns result in the final prediction.

Also pedestrians and cyclists are predictable based on the last few seconds. However a 100% confidential prediction will never be achieved because pedestrians almost have no constraints where to go.

Most related works like [45, 60] are based on object representations, similar to figure 1. So the precondition is that a map with all objects in the environment has to be provided. Based on this representation interactions are predicted between objects and their consequential trajectories. Nevertheless this kind of data is not every time available and is expensive to generate. Furthermore sensors like camera and lidar can only partially observe the environment because close objects hide everything what is behind them. For example when a car is left in front of the ego vehicle the sensors would only capture two sides of the object and the environment behind it is uncertain. However occluded objects are as important as visible ones for the ego vehicle so that objects have to be tracked. Dynamic object tracking includes the prediction of future positions for occluded obstacles so that just for creating a sufficient object representation prediction has also be learned.

Consequently a more efficient way would be to combine both tasks. Based on pure occupancy data that defines where an obstacle is detected, tracking and predicting would only be slightly different. This is why the following thesis deals with the prediction of multiple objects in a traffic environment for autonomous driving based on only slightly preprocessed sensor data for object recognition and tracking.

1.2. Proposed Approach

To tackle the task of multi-object prediction there are different machine learning approaches which have to be taken into account. Artificial Neural Networks have recently outperformed on multiple tasks like image recognition [40, 64, 66] and language understanding [74, 75]. But also in context of trajectory planning and prediction neural networks achieve impressive results [45, 72]. In order that a network can learn it has to be trained with a huge amount of data. For every input there has to be a corresponding correct output or at least a performance evaluation in order to determine how good the prediction was. Mostly creating the correct output requires human ability and takes lots of time. This is why the following work deals with training an artificial neural network on raw sensor data, including an occupancy and occlusion grid, without the need to expensively annotate them by a human.

This concept raises different challenges. On the one hand the expected output contains the whole objects' shapes. But this information is not available so that the network could not explicitly be trained on that. In addition to track objects every detected pixel would have to be assigned to an specific object. Yet this assignment does not exist and so the network has to cluster them by itself. Furthermore the recorded data is not perfect but very noisy. This challenges the network to decide which detection is real and which noise. On top of that the actual task has to be solved: prediction. Based on the past

few seconds a future trajectory has to be estimated for multiple time steps. Due to the fact that the data does not contain an object representation and so trajectories can not be assigned to certain objects, the task is changed to predict occupancy maps of the next few seconds. Vehicles are constrained to the road architecture so that this will be included as input to the system. The overall task is summarized in figure 2.

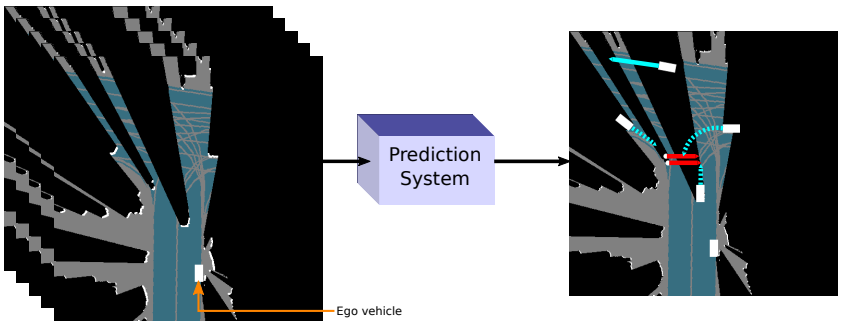


Figure 2: Multiple consequent frames, containing the occlusion as black, occupancy as white and the horizon map as background, are given to the prediction system as input. The output consists of the tracked and recovered objects including the proposed trajectory the would probably take (except of the ego vehicle for which that is already known). In the complex intersection scenario above the pedestrians (red) will cross the street while cars (blue) on the crossed lanes have to stop. However the car at the top can drive independently further. Objects besides the road are mostly static and therefore not considered yet.

The most related field to this problem is video prediction [19, 50, 80]. By observing raw video frames a network is trained to predict the next view time steps. For this the learning system has to analyze and understand the video including object and motion recognition. The correct output is given by the next frames of the video so that no effort is required to create them. All objects in videos are bounded to real-world physics that have to be handled by the prediction system [19]. However there is no annotation of this underlaying environment structure so that the algorithm has to learn it by itself. This task is not easy for the network depending on the complexity of the environment.
[20] has shown that future frames predictions can be generated action conditioned, so that a robot can test what would happen if it behaves in a certain way. This can be applied for the traffic environment prediction to test if for example a lane change is comfortably possible.

However all these approaches are focusing on unconstrained RGB images. For tracking and object shape recovering other approaches like Deep Tracking [58, 16] have to be considered too. Based on raw lidar data the learning network was able to regenerate the whole object's shapes and track them other multiple time steps. The training was similar to a prediction task so that it is easier for the network to learn the whole shape instead of the changing perspectives on objects.

The network, proposed in this thesis, is built up to do both tracking and prediction. The input consists of multiple images as described in figure 2, but the output is only the next occupancy map. To predict more than one future frame the output is fed back into the network to get the further predictions. Furthermore a new concept of a dilated dense block [32, 84] is developed to upscale the considered context for every pixel. This leads to a better recognition of interactions over huge distances.
In addition the predictions are conditioned on a certain input action. Hence a proposed trajectory could be tested and evaluated based on the corresponding environment predictions. Moreover the whole system could be used as a short-time simulator. However long-time sequences will not be usable for simulation purpose because the network is focused on predicting actions of the seen obstacles and not to create new objects.
Since the correct output is not available, different performance measurements are tested to train the system best. Finally the results of the proposed network are compared to state-of-the-art architectures.

1.3. Outline

The following thesis is divided into five parts. The first section introduces the area of machine learning and artificial neural network. Furthermore it summarizes related works to the prediction task that were taken into account. The overall model architecture and its relying datasets are described in the second section while the third includes the critical evaluation and testing of all approaches. In addition the results of state-of-the-art models are compared to the proposed ones of section three to show the improvements of the new architecture. At the end of this thesis an outlook of the further research and application is discussed in the fourth section and concluded by section five.

2. Foundations

This section gives a brief introduction to the field of machine learning and its application for prediction modules. The basics of general machine learning are explained in the first subsection, while the following one reviews artificial neural networks as a part of learning algorithms. As the last part of this section related works are presented and summarized.

2.1. Machine Learning

The field of machine learning summarizes algorithms that become more accurate in predicting outcomes without being explicitly programmed [42]. The basic concept is using statistical analysis on received input data and its corresponding correct output. But how exactly can an algorithm learn? A succinct definition for that is given by [51]:

> "A computer program is said to learn from experience E with respect to some class of tasks T and performance measures P, if its performance at tasks in T, as measured by P, improves with experience E." [51]

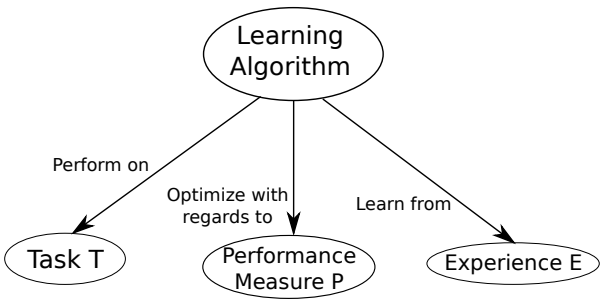


Figure 3: Overview of the relations between the learning algorithm and its task T, performance measure P and experience E. While T specifies the process, E defines how the datasets look like and P the way algorithm learns from the data.

The interaction between task, performance and experience is shown in figure 3 and described in the following sections. The layout is inspired by [24, 51] to give a brief introduction to the huge area of machine learning.

2.1.1. Task *T*

A learning algorithm is designed to solve a specific task. This task defines how an input should be processed and what the result of this algorithm looks like. As an example if a robot should be able to walk than the task it tries to solve is walking. The input could be a RGB image of its surrounding while the output is the leg's movement.

While the algorithm is learning on a task the input it gets is called an *example*. An example $\boldsymbol{x}$ is a collection of features that have been measured as a sample input of the

algorithm's system. Mathematically the example is represented as a vector $\boldsymbol{x} \in \mathbb{R}^n$ which elements are features. For an image this means that the pixel values are the features of the example.

After defining what a task is and how an input looks like they could be distinguished in different kinds. Due to the huge variation of possible tasks the following ones just represent a small subset of the most common machine learning tasks. For a more detailed list see [24].

Classification One of the basic tasks of learning algorithms is specifying the input's category among multiple possibilities. This means that the algorithm represents a function $f : \mathbb{R}^n \to \{1, ..., k\}$ that classifies the input example with its features to one of k categories. The output for choosing a category is usually defined by a numerical value $y = f(\boldsymbol{x})$ or an one-hot vector $y \in [0, 1]^k$ for which f calculates a probability distribution over classes. Typical use cases of this kind of tasks are object and handwriting recognition on images [44, 66].

Regression Some tasks require continual numerical values instead of discrete classes as output. In comparison to classification the represented function of the algorithm is defined by $f : \mathbb{R}^n \to \mathbb{R}$. Regression is used for various motor control tasks [54] and object localization in images [47, 63, 64].

Synthesis and sampling A different kind of task is when the algorithm is asked to generate new data examples that are similar to those in the training dataset. The input of this task is flexible and depends on the specific application. For predicting the next frame of a video sequence like [19, 80] the algorithm's function is represented by $f : \mathbb{R}^{n \times t} \to \mathbb{R}$ with t as number of input frames. The recent approach of GANs [25] for this task kind is discussed further in section 2.4.2.

2.1.2. Performance Measure *P*

In order a algorithm can learn it has to get a feedback of how good his predicted output was. So a quantitative measurement of its performance must be designed, usually specific to the belonging task T. In addition different algorithms can easily be compared using the same performance measurement. In the learning process the goal of the algorithm is to optimize its parameter towards performance gain.

A common way to measure the performance on tasks like classification is to define the accuracy of a model. Accuracy is the proportion of examples for which the model predicts the correct output. The opposite of it would be the error rate, the proportion of examples for which the model predict an incorrect output. For the task of regression where the output is specified by an continuous-valued number using this accuracy is not capable. A better method is the average log-probability or the mean-squared error

(MSE) that is defined as follows [24]:

$$\mathrm{MSE}(y, \hat{y}) = \frac{1}{m} \cdot \sum_{i}^{m} (\hat{y}_i - y_i)^2 \tag{2.1}$$

The advantage of this error function is that it increases quadratically with the difference of predicted output and correct data so that very large mistakes is penalized much more than many small ones.

However the penalty function often depends on the application of the system. To predict the next frame of a video sequence for example, MSE could be used because the pixel values are continuous, but this leads to blurry predictions. GANs [25] produce much sharper results because of a specific way for penalizing the generating algorithm but bring other disadvantages (see section 2.4.2).

2.1.3. Experience *E*

As the last part of the machine learning basics the source from which the algorithm learns should be explained. The basic component of experience is a dataset consisting out of a collection of examples. Mostly it is split up into a training and testing part. The training dataset is used for the algorithm to learn from while on the testing dataset the performance is measured regarding to new images which the algorithm has not seen for training. This prevents overfitting and tests the generalization of the system.

Three common different kinds of datasets could be distinguished depending on the way the algorithm gains experience of the data:

Supervised learning The easiest way for an algorithm to learn how the estimated function looks like is having for every example the correct output, called label or ground truth, and optimize it towards that. The structure of the ground truth depends on the task which should be learned. Figure 4b shows an example for classification, for which the label is only one value/class, and as comparison one for semantic segmentation, where the algorithm tries to classify every pixel by its own. Well known datasets for supervised learning are ImageNet [66] for image classification and Cityscapes [12] for semantic understanding of urban street scenes.

Looking from a statistical perspective supervised learning has a random vector $\mathbf{x}$ as input and learns to predict the correct label $\mathbf{y}$ by estimating $p(\mathbf{y}|\mathbf{x})$. The ground truth has to be generated by an instructor that is mostly human. While creating classification labels is quick producing semantic label for a high resolution image takes a long time. This is why datasets like Cityscapes [12] only have about 5,000 images whereas ImageNet [66] contains up to 14,000,000. Another application field for supervised learning is regression tasks like bounding box estimation on images [18, 46].

(a) Classification label (b) Semantic label

Figure 4: (a) The input is a RGB image like the one shown on the left [66]. The label belonging to it is given by the class "Firetruck". (b) For semantic segmentation the label is not only one class. It is an image of the same resolution as the input with a class label for every pixel. For visualization every class label is represented by a different color [12].

Unsupervised learning In unsupervised learning the ground truth is missing. The algorithm has to learn useful properties of the dataset structure without explicitly showing it. Usually for deep learning the aim is to capture the entire probability distribution either explicitly, as in density estimation, or implicitly, for generating new image at the task of synthesis and sampling. Another possible application after capturing the distribution is clustering the dataset in different categories by self-explored structures. So it learns a joint distribution of all features in the input vector $\mathbf{x} \in \mathbb{R}^n$ [24]:

$$p(\mathbf{x}) = \prod_{i=1}^{n} p(x_i | x_1, ..., x_{i-1}) \tag{2.2}$$

Although the distribution estimation and the way of learning looks different to supervised learning they both could not be clearly separated. Given the ground truth every unsupervised task could be converted to supervised, and on the other hand the conditional probability of the supervised way can be solved by learning the joint distribution of $p(\mathbf{x}, y)$ by unsupervised methods [24]:

$$p(y|\mathbf{x}) = \frac{p(\mathbf{x}, y)}{\sum_{y'} p(\mathbf{x}, y')} \tag{2.3}$$

A learn paradigm taking some of supervised and unsupervised is the semi-supervised learning where for example only some examples have label but not all of them. Unsupervised learning mostly takes longer until the algorithm has fully learned the distribution because it has to explore the structure by itself. But as a huge advantage no instructor is needed and so much more data is easily available. Applications of unsupervised learning are speech recognition/generation [67, 79] and video prediction [19, 80].

Reinforcement learning A common human way to learn new things is by "trial and error" [31, 76]. Simulating this behavior reinforcement learning is a paradigm in which the algorithm interacts with its environment and learns from feedback or

reward that is returned. Therefore the system selects actions in an environment in oder to maximize the expected reward. Figure 5 illustrates this framework.

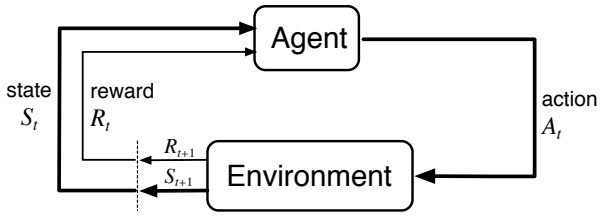


Figure 5: The learning algorithm is represented by an agent that takes an action A_t and receives a reward R_{t+1} from the environment. Its decision is based on the current state S_t and reward S_t. In games the state usually is a RGB image of the game the users sees and reward is given by the score [76].

An intuitive application of such learning problems is playing games [52, 37]. In classical computer games like Atari [52] there are a limited number of possible actions the player can take. The algorithm has to learn the expected reward of an action based on the input that is mostly the pixel values of the game. Approaches like AlphaGo [71] have successfully shown the possibilities of reinforcement learning while beating humans performance.
While games have always a certain reward signal (the simplest one is winning or loosing) some applications are missing that out. For this inverse reinforcement learning (IRL) is usable where an expert demonstrates the task that the algorithm should learn [1]. Reinforcement learning applies for computer games [52] and motion planning [72].

2.2. Artificial Neural Networks

Artificial neural networks are graph based models that are inspired by the human cognition [13]. A single neuron represents the smallest unit or node of such a network interlinked by many interconnections with other neurons using its activation to communicate. This behavior is simulated by simplified mathematical models for which the neurons are represented as graph nodes and the connections are weighted directed edges. To understand the mechanism of artificial neural networks the following section is divided into three parts. The first paragraph deepens the mathematical model of a neuron while the second handles with the basic architecture of a network and the third introduces the common convolutional neural network structure.

2.2.1. Artificial neurons

Figure 6 shows the mathematical model of a single neuron. It has multiple external inputs $\{x_1, x_2, ..., x_n\}$ which are usually the output of other connected neurons. All of

them are weighted by a corresponding parameter of $\{w_1, w_2, ..., w_n\}$ defining the relevance of a input for this neuron. Finally the weighted inputs are summed up to u for which an additional parameter b is taken into account. This variable is called "bias" and shifts the evaluated sum with an constant value. The neuron's output y is calculated by taking u as input for an activation function $g(u)$ that defines the output's mapping to the input. When a neuron learns it adjusts its parameter including input weights and bias towards the expected output.

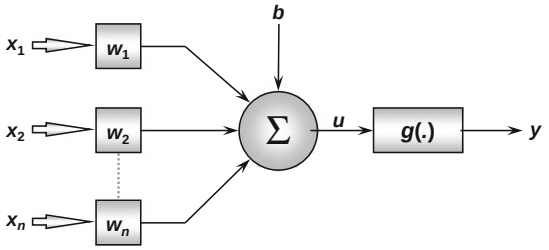


Figure 6: The mathematical model of a neuron consists of a weighted sum of its inputs that is shifted by a constant bias. The sum is given as input to an activation function to get the final output [13].

Summarizing an artificial neuron is represented by the following equation [13]:

$$y = g(b + \sum_i x_i \cdot w_i) \tag{2.4}$$

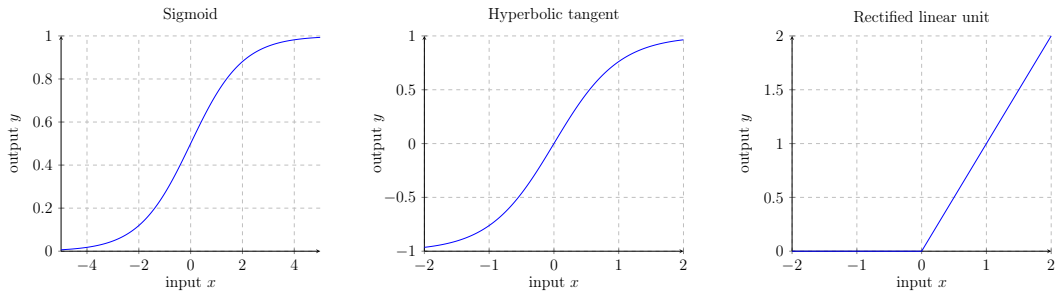
The activation function's goal is basically reducing the output range of values because the weighted sum u could theoretically be anything between $-\inf$ and $+\inf$. If many neurons are cascaded the output value could easily blow up. Furthermore from a biological point of view the output of a neuron is the activation that is also fixed to specific borders. There are several activation functions which can be used but only three should be described further here: sigmoid, tanh and ReLU.

The sigmoid function is defined as $\sigma(x) = \frac{1}{1+e^{-x}}$ and plotted in figure 7a. It is fully differentiable and nonlinear while having a range of values between 0 and 1. Due to it's output points are very steep near to 0 it tends to push all values to either 0 or 1. So it is a common activation function for the classification task.

Figure 7b shows the hyperbolic tangent that is defined as $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. It can be written by using the sigmoid function as $\tanh(x) = 2 \cdot \sigma(2x) - 1$ and shares most of the properties with the sigmoid function. Due to the output range of values is between -1 and 1 it is often used as output activation function for tasks like image generation.

Rectified linear unit (ReLU) is another more simpler activation function that is plotted in figure 7c. It is defined by the equation $\text{ReLU}(x) = \max(0, x)$ and so 0 for every

input lower than 0. The output range is $[0, \text{inf})$ what can still blow up. However one big advantage of this function is that it is less computational expensive and the activations are sparse what makes the network more efficient and faster to learn. ReLU is everywhere differentiable except at $x = 0$ but has no gradients for $x < 0$.



(a) $\sigma(x)$ (b) $\tanh(x)$ (c) $\text{ReLU}(x)$

Figure 7: (a) The sigmoid function maps the inputs to a range of 0 to 1 while having high gradients near to $y = 0$ to bring the output more to either 0 or 1. (b) The hyperbolic tangent is similar to the sigmoid function but has a output range of -1 to 1. (c) A rectified linear unit (ReLU) is 0 for all input lower than 0. All other values are processed linear so that they do not change.

Overall there is not the one activation function that can be used for every network. Moreover sigmoid is better to use for classification output, tanh for synthesizing and ReLU for image understanding inside the network.

2.2.2. Deep Feedforward Networks

After analyzing the concept of a single neuron this paragraph deals with the interconnection and combination of multiple neurons to a network. The quintessential artificial neural architectures are deep feedforward networks, or multilayer perceptrons [24]. These models are called feedforward because information flows through the whole network without any connections in which outputs of the model are fed back into itself. Networks with such connections are called Recurrent Neural Networks (RNN) and are described in section 2.3.

In general an artificial neural network consists of multiple layers that can be divided into three main parts:

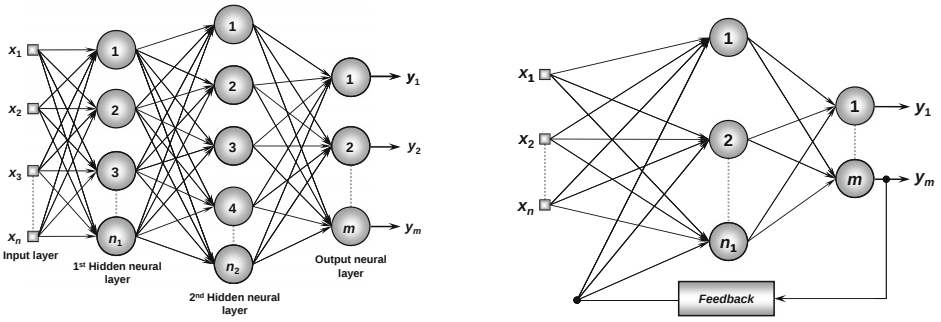
Input layer The first layer of a network receives information (data) from the external environment. Usually these inputs are normalized between -1 and 1 for better numerical precision for the mathematical operations and to keep the network's weight in a certain range [13, 24].

Hidden layers The main computing layers of a neural network are called "hidden layers" because they are invisible for the external environment. The structure consists of

multiple stacked neurons that are responsible for extracting patterns associated to the input. Neurons are only connected between layers but not within a single layer.

Output layer The last layer of a network is composed of neurons that are responsible for predicting the final output based on the previous hidden layers. Similar to the input of the network the output is usually normalized as well.

The structure of a simple feedforward network with two hidden layers is shown in figure 8a. The input is processed straight forward through the network and there is no recurrent connection. In comparison the model in figure 8b has a feedback from the last output layer back to the network. With this the next calculated output depends on the result of the previous step and is able to learn from a time component (for details see section 2.3).



(a) Feedforward Network (b) Feedback/Recurrent Network

Figure 8: (a) This figure illustrates fully connected feedforward network with two hidden layers. All neurons of one layer are connected to every neuron of the next layer with no feedback connection [13]. (b) In comparison to feedforward network a RNN has a feedback connection from one later neuron back to one in a earlier layer [13].

In the example of figure 8a all input nodes are fully connected with all neurons of the first hidden layer. Every neuron of this layer has therefore n inputs and n corresponding weights next to one bias that have to be adjusted by learning. With n_1 neurons the first hidden layer contains $(n+1) \cdot n_1$ learnable parameters. Layer two has therefore $(n_1+1) \cdot n_2$ and the output layer $(n_2+1) \cdot m$ parameters. With an increasing number of neurons the amount of parameters blows up what gets computational expensive and hard to learn. Sparser layers are for example convolutions which are described in the next paragraph.

2.2.3. Convolutional Neural Networks

A special kind of networks for processing data that has a known, grid-like topology like images, are Convolutional Neural Networks (CNN) [43]. They are based on the primary visual cortex of the brain and strongly inspired by neuroscientific research. CNNs

usually consist of two important layers: the convolution layer that is a variation of the mathematical operation called "convolution", and the pooling layer that reduces the inner width and height dimension of the data.

An example to motivate the usage of convolutions is having an RGB image of 64x64 pixels as input to an network. If the first hidden layer would consist of a fully connected layer with for example 1024 neurons, the needed amount of parameters would be $64 \cdot 64 \cdot 3 \cdot 1024 = 12,582,912$, and this only for the first layer. One way to reduce this number is by considering the geometry of the image. A pixel correlates much more with its close neighbors than with far off pixels [2]. Since the correlation of inputs is represented in the weights of a neuron the connections could supposedly be reduced to only a local region, called kernel, of the image, like 5x5 pixels (see figure 9). The stride of such an operation defines the step size over which the kernel is applied on the image. This leads to that the structure of the hidden layer is a similar grid to the input. The amount of weights for every neuron is reduced to 75 and overall 76,800.

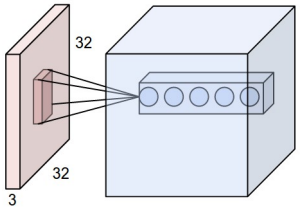


Figure 9: A neuron in a convolutional layer is only connected to a local region of the input. To apply different weight setting on the same kernel, multiple neurons are stacked behind each other which do not share the weights. However all neurons in one channel have the same to reduce the number of parameters [38].

Convolutional layers are going even further by sharing the weights between all neurons. So that one layer does not only consist of one weight set multiple neuron grids are stacked on each other which are called channels. For example applying a standard convolution layer with a 5x5 kernel and 64 channels on an input image of the size 64x64px, the first hidden layer would consist of $64 \cdot 64 \cdot 64 = 262,144$ neurons but only have $5 \cdot 5 \cdot 3 \cdot 64 = 4,800$ weight parameters.

The different kernels of a convolutional layer could be seen as filters like edge filter which are applied on the input. The resulting output has the same size as the input, but is not so rich of information anymore. To concentrate on the important results a pooling layer [42] reduces the height and width dimension by applying operations like max on all inputs of its kernel. It is worth mentioning that pooling is done on each channel separately. The max pooling operation is inspired by the biology where neurons that are strongly activated, exceeds their neighbors and weaken their influence. Figure 10 illustrates the mechanism of a max pooling layer.

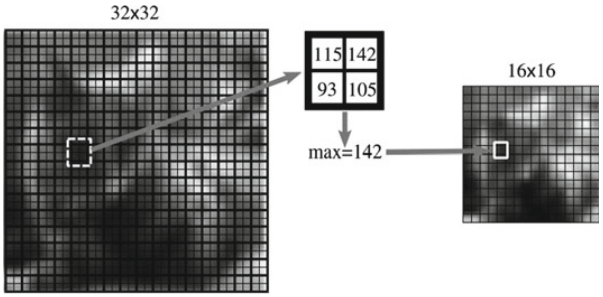


Figure 10: This figure illustrates the procedure of a max pooling layer with a kernel size 2 and stride 2. On every 2x2 pixel patch a max operation is applied to extract only the highest values. This reduces the dimensionality of the input by the factor of 2 [2].

The overall architecture of Convolutional Neural Networks consists of multiple stacked convolutions and pooling. Due to the output structure depends on the task of the network the final layers are adjusted to it. For example the task of classification ends up into a 1D-vector, so that the final layer usually is fully connected. In contrast synthesizing a new image requires a output of the same shape as the input, and therefore deconvolutions [56] are mostly used to upsample the dimensions.

2.3. Recurrent Neural Networks

Next to the Convolutional Neural Networks that are specialized for processing a grid of values, Recurrent Neural Networks (RNN) [65] are focused on processing a sequence of values $x_0, ..., x_n$. Of course such a sequence could also just be stacked together to one big input to a multilayer network, but this would not be parameter efficient. For example considering a model for language understanding that should extract a year out of a sentence. Two samples are "I went to Nepal in 2009" and "In 2009, I went to Nepal" [24]. The corresponding word appears in the first sentence at sixth and in the other at second position. Still the parameters could be shared between these two time steps.

The following paragraphs deepens the topic of RNN by describing how recurrence could be modeled in a neural network and presenting the Long Short-Term Memory (LSTM) [30] for long-time dependencies.

2.3.1. Modeling recurrence

Recurrent Neural Networks have, as in section 2.2.2 described, feedback connections from a neuron's output back into the network. The values of these connections are holden over time steps, so that processing input x_t depends on computation results of x_{t-1}.

A better understanding of a recurrent network is unfolding it over multiple time steps,

as illustrated in figure 11. Network A gets x_t as input to generate its output h_t. In addition there is a connection over time steps, so that x_{t-1} influences the prediction. Due to x_{t-1} is therefore influenced by x_{t-2} and so on, h_t depends on the whole previous sequence $x_0, ..., x_t$.

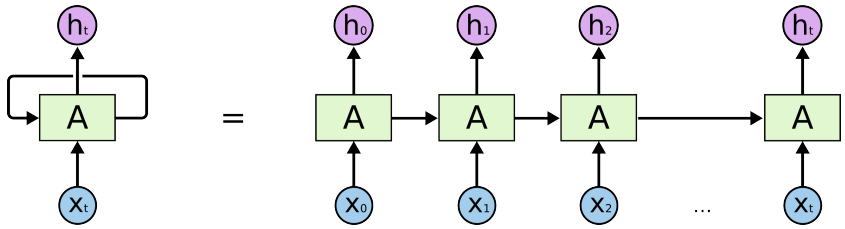


Figure 11: Recurrent Network A can be seen as a chain of multiple time steps with feedback connections between them. Due to it is the same network the same weights are used for processing the input and the recurrent output of previous step influences the final prediction [57].

The design and position of the feedback connection is based on the task. Some examples are:

- RNNs that produce an output h_t at every time step and have recurrent connections between hidden units as shown in figure 11.
- RNNs that produce an output h_t at every time step that is taken as input for the next time step. A common application for this design is video prediction [80, 19].
- RNNs that only produce an final output after reading a whole sequence. For this hidden units have recurrent connections in between to process and extract useful information out of every single input.

The detailed structure inside the network is not specified. Convolutional Neural Networks can be combined with RNN by taking the output of one convolution as input in the next time steps, like [58] for example did. However the computation of x_t strongly depends on x_{t-1}, but if long time dependencies are needed like for example in complex language understanding, other methods outperform for this task. One example are the Long Short-Term Memory Networks (LSTM) [30] that are explained in the next paragraph.

2.3.2. Long Short-Term Memory Networks

As [8, 29] have shown it is difficult to train recurrent neural networks to store information over a long time. Gradients flowing backwards in time tend to either vanish, what causes a need of training for very long time or does not work at all, or blow up, so that the weights are not stable anymore. To overcome these problems [30] proposed a much

more efficient gradient-based method called "Long Short-Term Memory" (LSTM).

LSTMs are a special kind of RNN which enable an constant error flow over multiple time steps. Between the time steps a cell state and the last output is passed through building up in a chain like structure (see figure 12). Inside the module four neural networks layers interact with each other to decide which information will passed further (remembered) and which will be added. The following paragraph describes this behavior more detailed using the name assignments of [57].

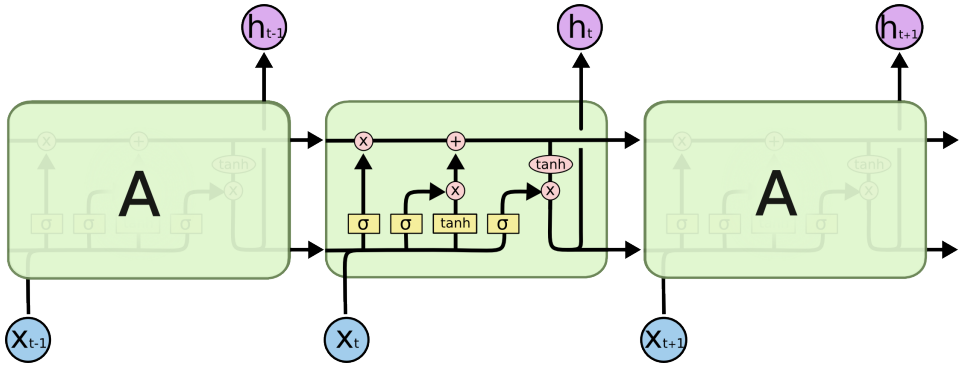


Figure 12: LSTMs are build up in a chain-like structure over time where they pass the cell state c_t and hidden state h_t to the next time step. The inner structure consists of a "forget gate layer", that selects which states should stay, "input gate layer", that chooses the new information which should be added to the states, and "output gate layer" for calculating the next hidden state [57].

The first step in a LSTM is the "forget gate layer". In here a mask of numbers between 0 and 1 are calculated in respect to the current input x_t and the last output/hidden state h_{t-1} to determine which information of the cell state c_{t-1} should be kept (respectively 1) and which to get rid of (respectively 0). Mathematically this layer could be written as follows [30]:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.5}$$

Next the module has to decide which information should be updated based on the current time step. This is done by a sigmoid layer i_t, calculating an update mask similar to the forget gate layer, and by a tanh layer to create the new cell state values $\tilde{C}_t$. Combining the mask and the new states with a point-wise multiplication to an "input gate" the cell state c_t can be determined in the following way [30]:

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \\ C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \end{aligned} \tag{2.6}$$

The last step is to calculate the "output gate" for the current time step based on the new cell state C_t. As the previous layers a filter is implemented by a sigmoid layer taking x_t and h_{t-1} into account and used on the cell state which was pushed through a tanh function to scale the values between -1 and 1. The final output of the LSTM is described by the following equation [30]:

$$\begin{aligned} o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ h_t &= o_t * \tanh(C_t) \end{aligned} \tag{2.7}$$

There are many variants on LSTMs. [23] adds "peephole connections" to every gate so that they also take the cell state into account. Another variation is the Gated Recurrent Unit (GRU) [11] which simplifies the LSTM a bit. The changes contain merging the forget and input gate into a single "update gate" and combining the cell state C_t and hidden state h_t.
The last type of LSTMs which should be mentioned here and is also used in section 3 is the convolutional LSTM (ConvLSTM) [70]. The standard LSTM uses fully connected layers for its inner structure. When using this module on spatial depending data like images it would have to be transformed into a single dimensional vector to be processed and looses because of this its spatial information. To hold these important information [70] proposes to use convolutions instead which has shown to handle spatial data much better.

LSTMs outperform on tasks where recognizing long-term dependencies is an essential key like speech recognition [67] and language modeling [74, 75]. The application of LSTMs are extended by using ConvLSTM so that the can also be used for video prediction [80, 81].

2.4. Related work

The following section briefly introduces related work which was taken into account for this project. First the framework DeepTracking [58, 16, 59] is summarized by focusing on the unsupervised training for keeping track of objects. The further part concentrates on generating the next frames based on a seen sequence. For this section 2.4.4 describes a model which tries to predict which pixel will move in which direction, while section 2.4.2 dives into the wide area of Generative Adversarial Networks [25]. As a example for this group of networks the Motion-Content Network [80] is explained deeper as it is used in section 3.

2.4.1. Deep Tracking

When the environment is discovered by only the robot's sensors it mostly can observe it only partially. Imagine a robot using a lidar to detect circular objects around it (see Figure 13). Through the sensor the robot can see the nearest objects to itself, but they can occlude other behind them. Still the occluded objects are as important to detect

than the visible ones, especially in the case of autonomous driving where occluded cars influences the ego vehicles behavior.

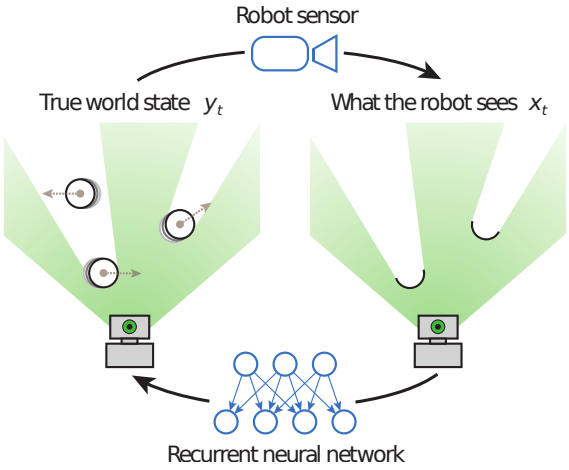


Figure 13: The robot can only observe the environment partially by its sensors. To recover also the occluded objects a recurrent neural network is used which captures the motion of objects to predict their next state [58].

To tackle this problem [58] proposes to use a recurrent neural network which keeps track on all objects including the occluded ones. The network's input is the environment which the robot sees. In Figure 13 it is represented as an occupancy image where a pixel defines if this position is occupied by an detected object or not. The output on the other hand contains all objects with their whole shape in the scene. For this task recurrence is needed in the network because the occluded objects could not be recovered by seeing only the current image.

The network's training could be done supervised but this would require labeled data. In real world application like autonomous driving there could no ground truth be captured by sensors. So a human has to generate it manually. However it could be hard to recover the whole track of an object being occluded. A better way to train the network is in an unsupervised manner which [58] proposes. Additional to the occupancy map the network will also get the occlusion map as input which comprises the information which pixel was seen by the sensor and which not. This allows the network to suggest where objects are occluded and to generate them there. Simultaneously when calculating the loss the same input occupancy map is used as ground truth while the occluded area must be taken out of account. Otherwise the network would be punished for generating occluded objects.

If the whole training procedure would be to give the network an occupancy map and calculate the loss on the same one as the input the network would only learn to push the input through without generating occluded objects. This is because the maximum likelihood function would be maximized at this point without learning complex patterns.

To counter this the network's input will be dropped out every N steps for further N steps by setting all inputs to 0. As a result the network has to learn the environment's patterns to predict the future occupancy maps. Further the network will not only learn to predict the occupancy map but also starts to track the occluded objects because it is easier to hold these in comparison to remove them when they enter the occlusion and regenerate when they are coming out again.

This concept is only a framework for any neural networks and does not require a specific architecture. So there are many possibilities to go further. For example extends [59] the task of tracking by adding a semantic segmentation while [16] attacks the problem of a moving ego-position. Section 3 takes up this idea and shows the application of DeepTracking for predicting the next states of a traffic environment.

2.4.2. Generative Adversarial Networks

Generating data like images is getting more and more important in the area of artificial intelligence as it can for example be used in cooperation with reinforcement learning algorithms for motion planning. The generative model predicts how the future states of the world will look like if the agent takes a specific action. Based on these generated states the agent decides what he should do and plans his next motions. For a recent example of such a model see section 2.4.4 and [19, 20]. However most models with standard methods suffer from blurring out the prediction over all possible states due to the loss functions like mean-squared error or ℓ_2 comes with the assumption that the data is drawn from a Gaussian distribution [50]. To predict realistic future frames the network has to concentrate on the most probable state and produce a sharper prediction. A new approach for achieving this are Generative Adversarial Networks (GAN) [25].

The framework of GANs introduces a minimax game between two players: a generative model G that tries to capture the data distribution p_{data} and produces new corresponding data, and a discriminative model D that tries to distinguish between ground truth and data that is generated by G. The aim for G is it to maximize the probability of fooling D so that it suggests the generated data of G's distribution p_g would be ground truth of the training dataset distribution p_{data}. Meanwhile D wants to detect the generated images as good as possible. A unique solution of this problem consists in G's distribution p_g capturing p_{data} and D being equal to $\frac{1}{2}$ for every generated and real data. Figure 14 illustrates this network framework.

The loss for training the GAN is based on the value function $V(D, G)$ which is used for the minimax game. Both models G and D are represented by a multilayer perceptron with parameters θ_g for the generator and θ_d for the discriminator. The generator's distribution p_g is learned by taking an input $\boldsymbol{z}$ distributed by $p_{\boldsymbol{z}}$ and using the differential model function $G(\boldsymbol{z}, \theta_g)$ to generate the output prediction. Due to G's aim is that D predicts its generated data is real the loss function consists of minimizing $\log(1 - D(G(\boldsymbol{z})))$ with D respectively as 1 for ground truth and 0 for predicted data. Simultaneously the

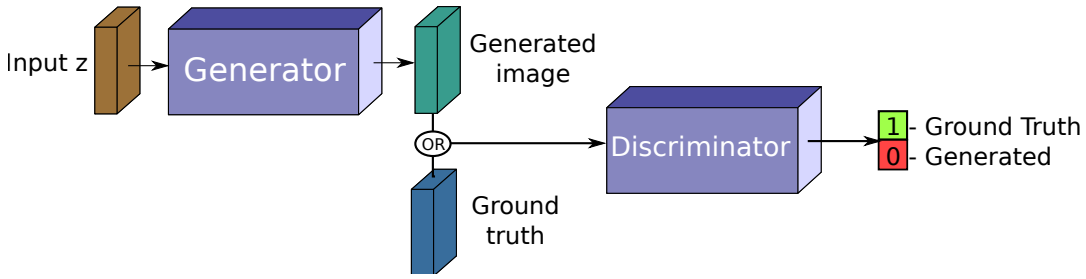


Figure 14: The generator G gets a random input $z \sim p_z$ and generates an image based on z. The discriminator's input is either a generated image by the generator or ground truth of the training dataset. Its task is to categories the input whether it is real or generated. Both of them are compete against each other because G tries D to predict ground truth for its generated data while D does the opposite.

discriminator function $D(\boldsymbol{x}, \theta_d)$ is trained to maximize the generators loss function and $\log(D(\boldsymbol{x}))$. Equation 2.8 summarizes this dilemma [25]:

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{data}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \tag{2.8}$$

However training a GAN is quite difficult because they are most likely not stable [4, 5, 27]. There are several variations of the GAN loss function to improve the stability of the system like training G to maximize $\log D(G(\boldsymbol{z}))$ to get stronger gradients early in training [25]. Another very popular improvement is Wasserstein GANs [5, 6]. Based on the *Earth-Mover* distance that provides a almost everywhere differentiable and continuous function under mild assumptions, the value function is changed to the following equation [27]:

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{data}(\boldsymbol{x})}[D(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{z} \sim p_z(\boldsymbol{z})}[D(G(\boldsymbol{z}))] \tag{2.9}$$

Even the stability is a big issue the framework of GANs is used very widely. The applications reach from learning disentangled representations in a completely unsupervised manner like rotation of an object [10] or creating paintings [77] and color images based on only a sketch [35, 85]. In the following section 2.4.3 the Motion-Content Network [80] will be presented as a GAN example for predicting future frames. Other similar works from which this work is influenced are a hierarchical prediction GAN [81] and a multi-scale approach by [50, 15].

2.4.3. Motion-Content Network

Images are a snapshot of the visual world. Compared to videos they are missing out the temporal component which provides a much richer description of the environment like motion and interaction between objects. So understanding and generating videos is a highly important but also challenging task. Not only there are so many variations

of possible object movements, further all interactions and motions depend on physical rules like how objects bump against each other.

Assuming a static camera position most of the frame's pixels are not changing comparing to the consecutive frame. Only a subpart contains dynamic objects that can move over time and have to be extracted by the generator to predict future steps. To simplify the task for the learning network a possible approach is decomposing motion and content of a given video and use them as two different inputs like proposed as the Motion-Content Network by [80]. The architecture as shown in figure 15 consists of two encoders processing the motion and content part of the video independently and converting it into hidden states that compress the extracted information. To keep track of the dynamic objects and understand their long-term behavior a ConvLSTM [70] is used to hold states over time. For the final prediction of the next frame both hidden state representations are concatenated and run through a decoder that results in a RGB image. To generate more than just the next frame the predicted output of the previous step is taken as input again to get the frame of the next time step. Even though this loop enables the network to generate an unlimited amount of future frames the image quality decreases over time. So the training is set to 10 predicted frames.

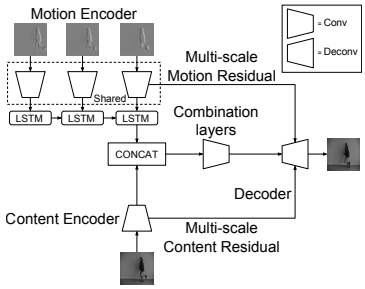


Figure 15: The Motion-Content Network processes the motion and content in two separate encoders which results are concatenated for the decoder [80].

Usually the dynamic objects are not clearly separated to the static pixels for using it as input to the network. So a human would have to label it. This leads to supervised learning which requires a huge effort by an instructor to create the ground truth. To prevent this the motion is approximated by the pixel's difference between two consecutive frames. Further residual connections [28] are used to improve the decoder's performance. The training procedure is based on GANs [25] combined with standard image losses proposed by [50]. This leads to a better stability of the GAN framework and generates more realistic images.

2.4.4. Learning object interaction through video prediction

When a object moves over time in a video it has a limited spatial space where it can be in the next frame. As well have the pixels a limited moving space assuming that

pixels only change because of its object movements. Based on this idea [19] proposes a model that explicitly models pixel motion and uses them on the previous frame. [19] constructed three different pixel advection models but the following paragraph only concentrates on the Convolutional Dynamic Neural Advection (CDNA) model which is the most object-centric approach and so most relevant for this work.

The CDNA network architecture, shown in figure 16, is based on multiple ConvLSTM [70] that are used to process the image. The output is not like for the MCNet [80] the next frame but 10 transformation kernels that represent the objects motion in the image. These kernels contain discrete pxiel distributions that are each applied to the entire previous image via a convolution. The results are 10 transformed images that save the expected value of the motion distribution for every pixel. Since objects usually only cover a subpart of the image the last layers of the network compute a pixel mask for every transformation kernel to compose the calculated values to one final prediction. In addition an eleventh mask is used for the original input image to weight the pixels that do not move at all. The masks sum to 1 at each pixel due to the values should not be increased unnecessarily.

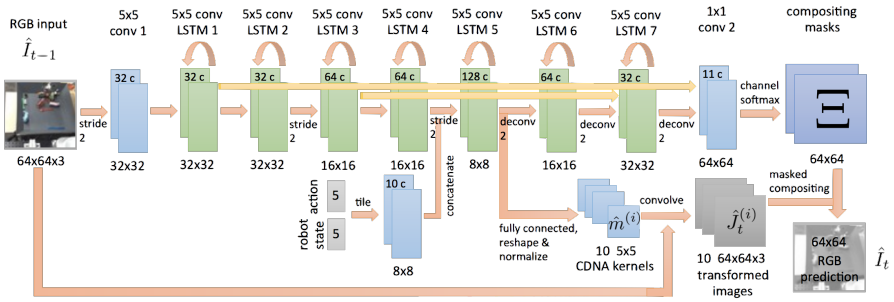


Figure 16: The CDNA model consists of multiple ConvLSTMs stacked on each other that process the input image and the robot's state and action concatenated with the hidden states. Output of this network are 10 normalized transformation kernels that are applied to the input image and composed according to the predicted weight mask [19].

In [20] the model was used for choosing the next action of a robot's arm. To make action-conditioned predictions the internal state like the robot gripper pose and action, for which the next frames should be predicted, are concatenated with the first smallest hidden states as shown in figure 16. Due to the state and action only are one dimensional vectors they are tiled over the hidden states resolution so that the shape fits for concatenating. As a result multiple videos can be generated for different suggested actions and used for choosing the best action to take.

3. Deep Prediction Learning

This section discusses the main contribution of this work. First the available datasets and their conversion is presented to describe the base on which the prediction system operates. After that the used network architecture for deep prediction learning is introduced separated into the structure, future step prediction and loss calculation. This model will be evaluated and tested in the following section 4.

3.1. Environment datasets

For prediction learning it is essential how the environment is represented. This requires first of all a system architecture which can observe and record the surrounding. Then the detections have to be processed to get a more capable format for prediction system like figure 17 shows.
The first paragraph takes up the system setup of an autonomous driving vehicle that is similar to the one recorded the data used in this work. Second the general representation format is described and how it can be generated. The task specific changes of this format follows as a third part. In the last paragraph "motion maps" are introduced as a new richer road map for prediction concerned tasks.

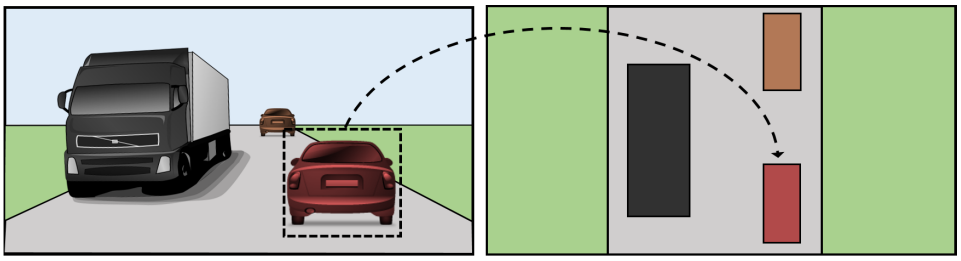


Figure 17: Images taken by the camera have a 2D representation where the depth information has to be inferred. A better way is using the bird's-eye view for which every vehicle has to be matched to a certain position [61].

3.1.1. System setup

For a better understanding of the essential components of an autonomous vehicle and data generation, the important sensors and system architecture are introduced in this section. The work from Urmson et al. [78] and Ziegler et al. [86], whose architecture is shown in figure 18, serves as orientation. The following list contains the most important sensors:

- **Lidar** (acronym for **Li**ght **d**etection **a**nd **r**anging) is an active optical sensor which sends out laser beams and evaluates their reflection from the surroundings. Based on these reflections, a 3D visualization fo the environment can be generated.

- **Cameras** can receive color information compared to Lidar and Radar. They are also one of the cheapest and most available sensor. Bertha [86] is equipped with two large field of view cameras, one in the front and one in the back (see figure 18). In addition it uses stereo cameras to extract 3D information of the surrounding.

- **Radar** (acronym for **Ra**dio **d**etection **a**nd **r**anging) uses radio waves to detect velocity and range of other objects. It is already widely used for adaptive cruise control and has different ranges depending on the use case. This is why Bertha [86] has multiple short and long radars to cover all areas around the car in different ranges. Figure 18 visualizes this structure.

- **GNSS** (short for **g**lobal **n**avigation **s**atellite **s**ystem) uses satellites to provide vehicles with their global location (latitude, longitude, elevation). The resolution is within the range of a couple a meters, but can be improved by using inertial measurement units (IMU) that further measures the acceleration and orientation of the car [78].

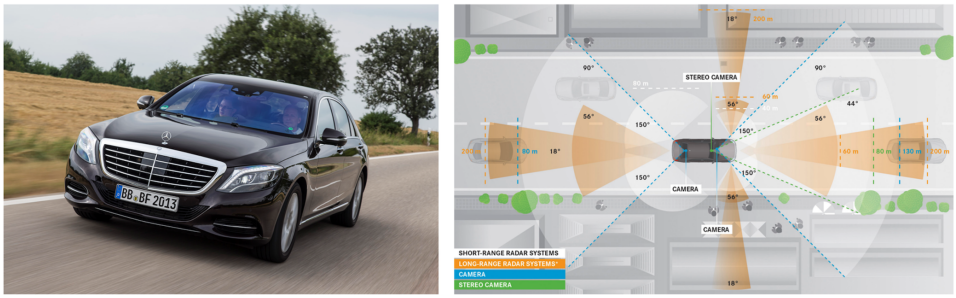


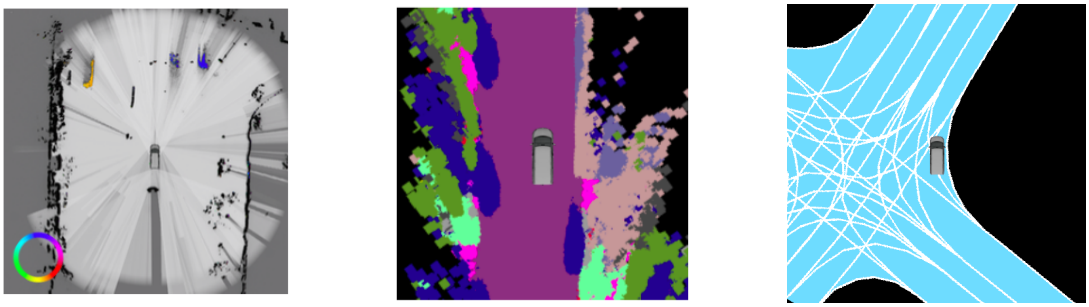
Figure 18: The Bertha Benz experimental vehicle and it's sensors. The radar's field of view is shown in orange and the camera's in blue. Together the sensors cover the whole surrounding [86].

3.1.2. Gridmap

The raw sensor's input have to be processed and analyzed to generate an environment representation on which the future trajectory of the vehicle can be planned. The approach that is used for this work is fusing different sensors together to a grid-map based image with multiple channels having a bird's-eye view on the surrounding and centered ego vehicle. The sensor's output are first preprocessed independently and combined on different channels in the gridmap. Following maps are provided:

- The **velocity map** captures all moving objects and their orientation. To encode the direction efficiently in an image, so that for example Convolutional Neural Networks can process them efficiently, a color wheel as shown in figure 19a is used to represent the objects orientation. It is generated by using Radar and Lidar.

- Static objects are included in the **occupancy map** that is added as black objects in figure 19a. Mainly Lidar points are used to generate such a map.
- The **semantic map** determines the type of an object. For this camera images are processed by a combination of Convolutional Neural Networks and classical computer vision algorithms [21]. To improve the map Lidar and stereo camera could be used as well. The different object classes are represented in multiple colors in figure 19b.
- The road structure is encoded by the **horizon map**. It is a cropped image of a high resolution map showing the road and lanes (see figure 19c). Different sensors are used to localize the ego vehicle as accurately as possible.



(a) Velocity map (b) Semantic map (c) Horizon map

Figure 19: (a) Dynamic objects are encoded by their velocity orientation using a color wheel (see bottom left). In this figure the static objects are represented by the black color to summarize all detected objects. (b) The semantic map classifies pixel in one of multiple object classes using different colors. For example constitutes purple the class road and blue cars. (c) The horizon map defines the structure of the road to get an overview where the vehicle can drive.

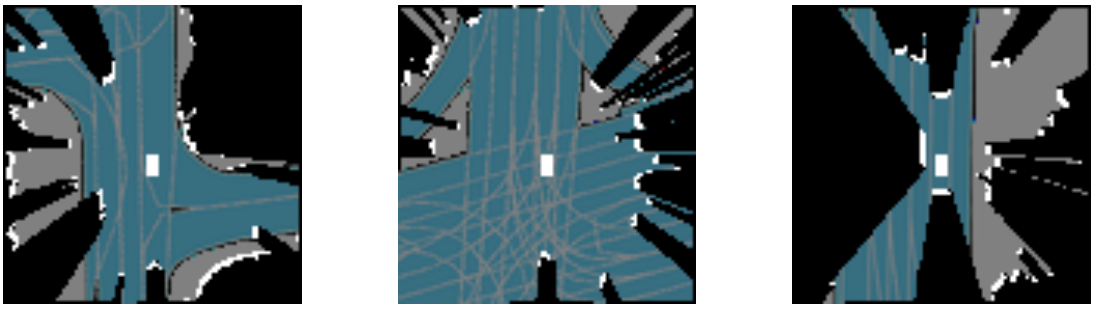
3.1.3. Tracking dataset

For the task of prediction learning the gridmap channels have to be adjusted. Due to the goal is track objects similar to [58, 16] an occupancy map that determines whether a pixel is occupied or not, and an occlusion map to get a mask for the gradient flow, has to be generated.

To get this the gridmap velocity and occupancy map is converted to a boolean map that is 1 for pixels being occupied and otherwise 0. The ego vehicle is added as rectangle in the center of the map because other objects behavior strongly depends on it. The occlusion map is generated on the base of the new occupancy map by simulating lidar beams in different angles. If a beam hits an obstacle, so a pixel with the value 1, every

pixel in the same angle behind it is occluded.

However it has to be considered that the lidar sensor's point are noisy and recorded in 3D compared to the 2D gridmap. This is why the sensor detects points in occluded area but these are not the all pixels of the corresponding object. So the occlusion map is used to mask out the detected occupancy in occlusion. Figure 20 shows some resulting frames of this procedure.



(a) Sample frame (b) Intersection (c) Close objects

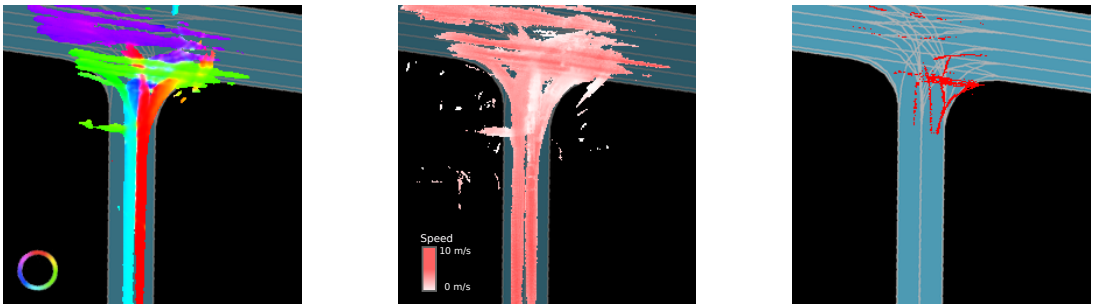
Figure 20: (a) Sample of the dataset. The objects on the road are cars occluding everything behind them. The horizon map is added for orientation. (b) Example of a intersection. Because of no closeup objects the map is quite free of occlusion. (c) Close objects occlude big parts of the map. Especially when the ego vehicle stands at a section surrounded by many cars most of the map is occluded.

3.1.4. Motion maps

The horizon map only shows the lane's structure on the road but not the driving direction of the vehicles. However this information can be added by simply recording a long sequence of velocity maps at the corresponding positions. But there is even more that can be inferred by observing the traffic. The map contains static information about the maximum speed, stop lines and traffic lights while for example traffic rules are not considered. Human drivers however follow these rules so that they can be observed and inferred by only the traffic.

This idea is implemented in "motion maps". Figure 21a shows the result of recording 45 minutes traffic at one intersection. The orientation map clearly demonstrates which lane has which driving direction. In addition the crossing of colors indicates a crossing of traffic. So a stopping is required by one of the lanes if there are cars on both. To detect which lane has the right of way the velocity intensity map in figure 21b records the average speed of an object at every pixel. Stopped vehicles on the road are separated from static objects by comparing the semantic map with the occupancy map. As a result

the lane heading to the top has to stop very often at the intersection. So the crossing lane must have the right of way. Furthermore the motion maps detect a parking lot entrance/exit on the left which was not included in the horizon map. Overall the motion maps can also improve the certainty of the horizon map.



(a) Orientation map (b) Intensity map (c) Pedestrian map

Figure 21: (a) The velocity orientation is encoded using the color wheel at the bottom left corner. Every lane can now be assigned with a velocity direction, and even crossings and turns are visible. (b) The intensity map shows the average speed of objects encoded from red as fast (10 m/s) to white for stopping vehicles. This motion map infers that the crossing traffic has right of way compared to the lane heading to the top. (c) The pedestrians trajectories are visualized by a red line on the horizon map. Due to some riders are misclassified as pedestrian there are some points in the middle of the street.

Not only vehicle patterns can be observed. The map does not contain any pedestrian crossings for an intersection but there are certain patterns which can be detected. Figure 21c visualizes all pedestrian trajectories that were seen in the 45 minutes sequence. Due to the semantic is not perfect and the classes “rider” and “pedestrian” are quite similar the map also detects some people in the middle of the road which can be ignored for the further evaluation.
The bottom right corner of the intersection stands out by analyzing the trajectories. Looking at the occlusion map there was an object that occludes the pedestrians when they arrive to the intersection. With knowing this fact the autonomous car could slow down in term of expecting a pedestrian might come out at this position. However the recording and generation of such motion maps is expensive. So only small experiments on standing situations could be tested in section 4 but motion maps for moving scenarios will be more a long-term goal.

3.2. Learning predictions on moving ego vehicle

The following section introduces the proposed network architecture that is trained on the previous datasets. There are multiple challenges that the model has to handle:

1. The input is recorded data by real-world sensors. This leads to a noise distribution that has to be flattened by the network.
2. The data is strongly biased on free space. However it is even more important that the network predicts all occupied pixels correctly.
3. Every object can interact with each other. This is why the network has to take every input pixel for the whole output into account.
4. Sensors like radar and lidar can only detect the first obstacle its beams hit. Nevertheless other objects can be occluded by them and have to be considered as well for the prediction.
5. The ego vehicle is moving so that all objects' motion are relative to its speed. The interactions are still based on the absolute speed that has to be extracted.
6. When the system has to predict multiple frames into the future, new objects can enter the sensor's field of view. Some of them are inferable and therefore can be learned.
7. To apply the network in the car it has to run in real time. This requires a run time that is less than 200ms.
8. All these challenges have to be combined and weighted to one performance measurement.

All these problems are considered in the following approach. To introduce the new model the section is separated into three parts. First the network architecture is described and explained in detail. Paragraph two deals with the usage and framework in which the network is embedded. The section closes with a closer look at different loss functions and their application in training.

3.2.1. Network architecture

The input to the network consists of the occupancy and occlusion map similar to [58, 16]. In addition all objects in a traffic environment are constrained to the road structure. This is why the input is extended by a road map that is 1 for every pixel on the road and otherwise 0, and a lanes map that represents the lane structure on the road. Furthermore to get action-conditioned predictions like [19, 20] the ego vehicles state and action have to be considered by the network. The corresponding vector to this consists of 5 properties: horizontal and vertical velocity and accelerations as well as the yaw rate. The position of the ego vehicle is always the same in the gridmap and has not to be obtained by the given vector. Output of the network is the predicted occupancy map one time step in the future.

The network design itself is inspired by [19] and shown in figure 22. It consists of an encoder for extracting the important features out of the input, and a following decoder

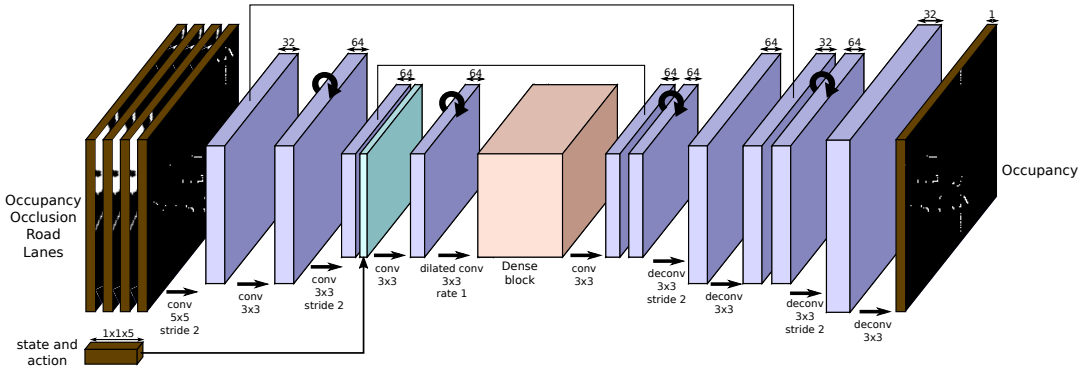


Figure 22: The proposed network consists of an encoder-decoder structure with four ConvLSTMs [70]. The feature maps are represented by the blue rectangles on which different layers are applied. The ego vehicle's state and action are added as additional channels to the third feature map and a dense block is used on the lowest scale to maximize the receptive field. In addition residual connections [28] and deconvolutions [56] are used in the decoder which ends up in a one channel prediction of the input resolution.

for generating an output image out of the encoded features. The encoder's first layer is a convolution with kernel size 5x5 and stride 2 to capture low level features and reduce the input dimensionality. Directly after that the first convolutional LSTM (ConvLSTM) [70] is applied to hold low-level features over multiple time steps. Followed by an additional 3x3 convolution with stride 2 to further reduce the height and width dimensions the state and action of the ego vehicle is added as extra channels to the network. As proposed by [19] the input vector is tiled over the width and height dimensions of the corresponding feature map to concatenate them easily. A following ConvLSTM tracks these combined features over time.

So far a neuron has an overall receptive field of 15x15 pixels on the input. As [48] has shown that the corresponding effective field, meaning the field of pixels with a notable impact on the neuron's output, is much smaller and based on a Gaussian distribution it can be considered that the network until this layer has analyzed every object by itself. To predict the future position of all objects their interaction has to be taken into account and therefore a much wider receptive field is needed. The selected way of efficiently doing that it adding a dense block [32] but replace the standard convolutions by dilated convolutions [84].

A dense block consists of multiple convolutions connecting each layer to every other layer in a feed-forward fashion [32]. Having for example four layers, then layer one is only connected to the input, layer two is connected to the input and output of layer one, layer three is connected to the input and output of layer one and two, and so on. This leads to a huge input channel size and parameter amount for the last layers. To reduce

them 1x1 convolutions are used as bottleneck layers before applying the convolution with a higher kernel size.

However this does not increase the receptive field strongly because it only takes standard 3x3 convolutions. Another layer that supports exponential expansion of the receptive field are "dilated convolutions" [84]. The kernel with rate n of a dilated convolution differs from a standard vanilla convolution by the fact that only every n^{th} entry of the feature map is used. This means that when a 3x3 kernel with a dilation rate of 2 is applied on a feature map every second entry is only taken into account. Cascading such dilated convolutions leads to an exponential expansion of the receptive field but further looses local information by increasing the number of stacked layers.

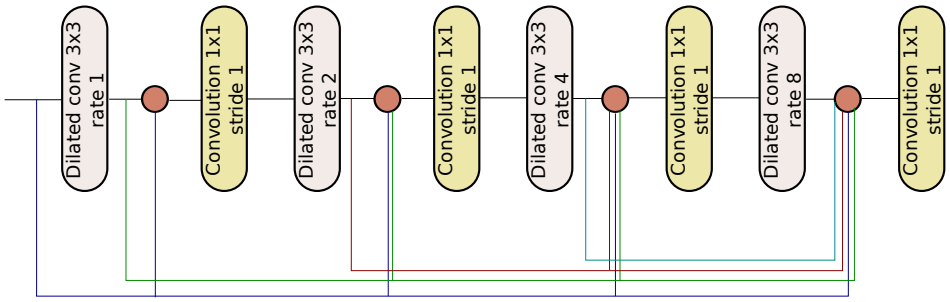


Figure 23: The figure shows the structure of a dense block [32] with four dilated convolutions [84] increasing their rate exponentially over depth. All layers are connected with each other in a feed-forward manner while 1x1 convolutions are used for reducing the channel size.

To still hold the local information but also expand the receptive field the approaches of a dense block and dilated convolutions could be combined. The standard vanilla convolutions in the dense block are replaced by dilated convolutions with exponentially increasing rates to achieve a wide receptive field. For the prediction network the depth of the dense block is set to four layers as shown in figure 23. To further reduce the input channel size to a fixed number of 64 a bottleneck layer is applied before every convolutional layer. The output of the whole block is the concatenation of calculated features from all layers reduced by another 1x1 convolution. So the local features are still given by the output of the dilated convolution with rate 1, while the wide context is considered by the output of the last dilated convolution.

After such a dense block the network architecture is continued with a third ConvLSTM. From here the decoder of the network begins. It consists of multiple stacked deconvolutions [56] that are build up in an inverted manner compared to the encoder. Furthermore residual connections [28] are used obtain low-level features for decoding. The final output is a one channel deep occupancy map of the same resolution as the

input. The detailed architecture can be viewed in figure 22.

To achieve good results with a network it is important to normalize it. The input is shifted from 0 and 1 to -1 and 1 as well as the output. Standard methods for inner normalization like batch normalization [34] and layer normalization [7] require huge memory and computational time what effects the training process. A newer approach is using "scaled exponential linear units" (SELUs) [39] as activation function which have been proved to be self-normalizing in combination with a certain range of weight initialization. This requires no additional memory and beats networks using batch and/or layer normalization [39]. So for every convolution the standard ReLU activation function is replaced by SELUs within the network architecture.

3.2.2. Future step prediction

After defining the network architecture in the previous section it has to be considered how this network could be applied and trained for predicting occupancy maps on a moving ego vehicle.

The simplest structure is feeding the output back as input as it is done by [19, 50, 80]. Several problems occur for this strategy that are compensated by the following mechanisms and summarized in figure 25:

Prediction occlusion map The output is only the occupancy map of the next step. Even though the road and lanes map could be calculated and determined based on the given ego vehicle speed and yaw rate before training or parallel to the network's execution, the occlusion map has still to be computed after the prediction. This usually is computational expensive (approximately 10-15 milliseconds) and in addition not differentiable. An alternative approach is given by [58] using a occlusion map of "everything is occluded" for future prediction. Not only that it needs no computation time at all, it further helps the network to learn the task of tracking objects. However with this concept the network can always distinguish between given ground truth as input or its own generated frame and adapts its prediction to that.

Input noise The networks prediction contains noise due to its output range of values is continuously between 0 and 1. If a noisy image is taken as input to the network the noise usually increases on the output. The uncertainty of a prediction gets higher when the input already is uncertain because the network's output has a standard derivation which will never be zero. Transfered to the problem of generating occupancy maps this means that when for example at $t = 1$ an object is predicted at a certain position with a probability of 0.9 (so that the output value of its pixel is 0.9) and the generated image is taken as input for the next time step, the object probability will drop to 0.81 or similar due to the network adds its uncertainty again. Doing this over multiple time steps the object's probability

will decrease and finally vanish. For free pixels this is not the case because 90% of the occupancy map itself are not occupied and so it is a lower cost to predict 0.

To prevent the vanishing of objects a "keep-alive function" could be applied on every output before taking it back as input again as shown in figure 25. This function is for example $f(x) = \tanh(a \cdot x)$ with a factor a that is greater than 2 for an input in the range of -1 and 1. As plotted in figure 24 it pushes the values in between more to the edge points. With such a keep-alive function the image noise is reduced so that the input gets sharper. Different function equations are tested in the experiments of section 4 to prove their ability.

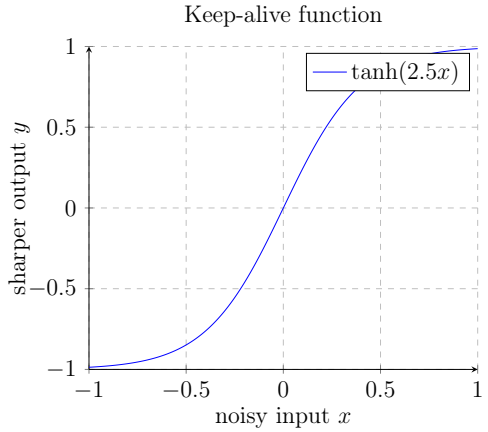


Figure 24: A sample keep-alive function $\tanh(2.5x)$. Inputs between -1 and 1 are pushed toward the edge points to reduce the inner noise.

Ego motion When the ego vehicle moves the predicted objects motions have to be with regard to the motion of the ego vehicle because it is centered in every frame. Even though the actions of the ego vehicle are an additional input to the network it is still hard to predict this correlation. Imagine next to the ego vehicle two cars are driving and then the ego vehicle slows down and stops. The output would look like the two cars are highly speeding up although they are not changing their velocities. Also all static objects around the ego vehicle would move inverted to its motion so that the model has to track much more objects.

A strategy for compensating the ego motion is transforming the output images with regard to the ego vehicle velocity and yaw rate like [16]. Standard 2D affine transformation functions are not differentiable so that a Spatial Transformer Module (STM) [36] is the better solution. The STM applies a point-wise transformation on the image as follows [36]:

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \mathbf{A}_\theta \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix} \tag{3.1}$$

Usually the transformation matrix $\mathbf{A}_\theta$ is learned by a localization network but in this case the matrix is already defined. Given Δt as the period duration of a frame, γ as yaw rate and v as the velocity, the transformation matrix is approximately determined by [22]:

$$\mathbf{A}_\theta = \begin{bmatrix} \cos(\gamma \cdot \Delta t) & -\sin(\gamma \cdot \Delta t) & 0 \\ \sin(\gamma \cdot \Delta t) & \cos(\gamma \cdot \Delta t) & -v \cdot \Delta t \end{bmatrix} \tag{3.2}$$

Not only the output depends on spatial information. Also the hidden states have to be transformed regarding to the ego motion so that the model can still track objects. Summarized STM is applied on the output and all hidden states of the network, as in figure 25 shown by the red components. With this strategy the prediction task is the same for standing and moving ego vehicle for the network so that it is easier to learn and generalize.

Punishing unseen objects The future states of a traffic environment depend on the previous ones. So the network has to see a short frame sequence before it can start to predict the next occupancy maps. This is done by giving the ground truth as input and the output is only used for loss calculation. The task is broken down to the prediction of one frame. But even when for example 10 ground truth frames are shown, the future environment is not completely determined with that. New objects can enter the ego vehicle's field of view after the ground truth sequence and could not be predicted for certain. Still the network is punished for not predicting these. However this situation is unpreventable because it can happens within the next frame so that even showing ground truth does not eliminate the probability of unseen objects in the prediction label.

As a conclusion a consideration between training on predict as many future frames as possible what improves the generation quality due to the last frame depends on all previous frames, and a low probability of training on unseen objects which could lead to hallucination of entering cars, has to be found. One way to go is to alternate between showing n ground truth frames and predicting m future states. This technique is also applied in [58] because for Deep Tracking it is as well absurd to train on unseen objects but still want to have future steps prediction. Figure 25 shows this mechanism for $n = m = 2$. Alternatively the network can be trained on predicting unseen objects but this seems to be a even harder challenge.

Overall the framework combines all of the strategies as shown in figure 25. When the model is trained it gets alternatively a certain number of ground truth frames and then

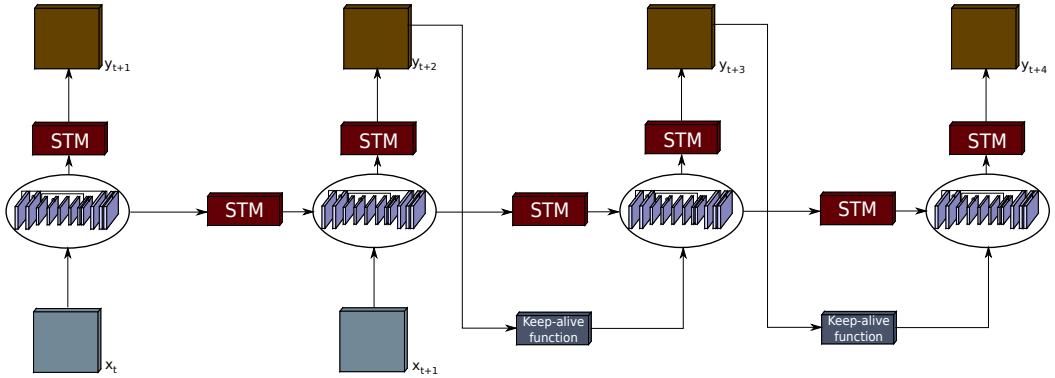


Figure 25: The training sequence alternates between giving $n = 2$ ground truth $x_{t+l:t+l+2}$ and $m = 2$ prediction frames $y_{t+l+2:t+l+4}$ as input. While predicting further time steps a keep-alive function is used to reduce the input noise. The output and all hidden states of the network are transformed by a Spatial Transformer Module (STM) [36] based on the ego vehicle motion.

predicts several recurrently. While predicting next frames the occlusion map represents a fully occluded environment and a keep-alive function is used to get rid of some output noise. All generated frames are taken into account for the loss calculation with which the following section deals. To compensate ego motion Spatial Transformer Modules are applied on the output and all hidden states between time steps corresponding to the velocity and yaw rate of the ego vehicle.
If the network is fully trained and should be tested, the challenge of unseen object prediction can be neglected because the network's parameters are not changed anymore. So the procedure switch to showing n ground truth frames and predict the needed amount of future time steps which can be much higher than m used in training. The mechanism of alternating between ground truth and prediction does not work for testing because if the ground truth of future frames would be known then the system would not be needed anymore.

3.2.3. Loss calculation

As described in section 2.1.2 the performance measure that is represented by the loss is one of the most important parameters of the training process. It defines how the network should optimize its parameters based on the comparison between prediction and ground truth.

Due to the ground truth of the occupancy maps is a classification of "pixel is occupied" and "pixel is free" a loss like cross entropy fits much more likely than regression losses like MSE [24]. Further to get realistic predictions and reduce the input's noise while putting the output back as input the GAN framework [25] as described in section 2.4.2 is a possible way to do it. As counterpart an additional loss function for pushing the network to sharper predictions is tested. The following paragraphs deepen these components of

the loss calculation:

Binary cross-entropy (BCE) The cross-entropy function is defined by

$$C = \frac{1}{N} \cdot \sum_x [-y \ln a - (1-y) \ln(1-a)] \tag{3.3}$$

where N is the number of items of the training data like the amount of pixels in an image, x is the input, y is the corresponding ground truth to x, and a is the prediction/output of the network for x [55]. In a classification task the ground truth y is either 0 or 1 and the prediction a is also between 0 and 1. Given these ranges of values for y and a the loss is also called "Binary cross-entropy" (BCE) [24]. To understand this loss function it is easier to split it up into two parts:

- $-y \ln a$ is the loss for items that's corresponding ground truth is 1. For these the loss is defined by $-\ln a$ while it is 0 for items with a ground truth of 0. Furthermore the loss is heading towards infinity for wrong predictions so that the network is trimmed to predict 1.
- On the other side $-(1-y)\ln(1-a)$ is the loss for ground truth 0. Its loss $-\ln(1-a)$ is compared to $-\ln a$ mirrored along $x = 0.5$ so that losses for 0 and 1 are weighted the same.

Furthermore cross-entropy avoids the problem of learning slowing down especially when using sigmoid or *tanh* as activation function [55]. Because the gridmap occupancy map as seen in section 3.1.2 consists of an inequivalent distribution of occupied and free pixels. So the loss function concentrates more on predicting correct free pixels than the occupied ones although for the task it is more important to predict the occupied areas also as it while suggesting a free pixel to be occupied is not so critical. This is why a weighted binary cross-entropy is used for training the network architecture. However the loss for predicting a free pixel as occupied would be still heading towards infinity. To prevent this the logarithm is slightly stretched so that it has a smaller maximum in the range of 0 and 1.

The concrete parameters which are chosen based on testing a set of possibilities are weighting the loss of 1s by 2 and stretching the logarithm for the loss of 0s by 0.9. Summarizing this modification the loss function is defined as follows:

$$C = \frac{1}{N} \cdot \sum_x [-2 \cdot y \ln a - (1-y) \ln(1-0.9 \cdot a)] \tag{3.4}$$

GAN Discriminator Even if a keep-alive function as described in the previous section helps to sharpen the predictions between time steps the network itself is not motivated to do so. GANs are a common framework to achieve with a discriminator loss very sharp predictions by punishing the model if there is a clear difference

between ground truth and prediction [26, 25]. The generator G is represented by the proposed network of section 3.2.1. Because the plausibility of the occupancy map predictions is strongly depending on the time sequence, for example that a occupied rectangle will not be vanishing within one single time frame, the discriminator network has to be recurrent as well. Still the occupany map is quite simple compared to an RGB image. So the architecture of D is reduced to four convolutions and one final fully connected layer to get the prediction, as it is shown in figure 26. The last convolution is build up by a ConvLSTM [70] to track only high-level features.

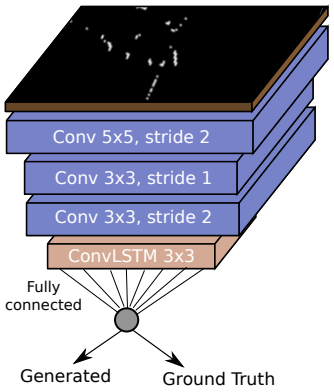


Figure 26: The recurrent discriminator consists of four convolutions which are applied on every frame. The last convolution is implemented recurrently with a ConvLSTM [70] so that it can captures changes over time. Similar to the standard GAN framework the final layer is a fully connect one compressing the features to a prediction of "generated" or "ground truth".

The generator gets ten ground truth frames to see so that it can learn the context of the given sequence. This method is also applied for the discriminator while seeing ground truth or generated frames afterwards. So based on only the first frames the discriminator will not be able to tell whether it is ground truth or prediction because the input is in both cases the ground truth. In addition the generated frames will get worse over time and so will be easier to spot later for D. This is why the discriminator will not be trained on all seen frames but only the last fews.

Sharpening loss Discriminators are used to extract complex blurriness of RGB images. Due to predicting the occupancy map is not a regression but a classification task the sharpness could easily measured without any additional network. The only possible labels are 0 and 1, so that predicting anything else leads to blurry output which can be punished by a loss function.

The simplest function that fits to this task would be a linear one like :

$$L(x) = \begin{cases} x, & \text{for } x \leq 0.5 \\ 1 - x, & \text{for } x > 0.5 \end{cases} \tag{3.5}$$

. The problem with such a linear function is that it is not differentiable at the point $x = 0.5$. A better suggestion would be to take a parable as defined by $L(x) = -(2x - 1)^2 - 1$. Even it is everywhere differentiable the loss gradients are at the two node points 0 and 1 at a local maximum. This would push the networks output strongly to 0 and 1 although the sharpening loss should only separate the predictions clearly to free or occupied while the classification loss like BCE should be superior near to those points.

A loss function which fulfills all the given requirements is plotted in figure 27 and given by

$$L(x) = [(2x - 1)^2 - 1]^2 \tag{3.6}$$

. It has a maximum at 0.5 due to there is the highest uncertainty and blurriness while decreasing to both sites. The gradients are 0 for $x = 0$ and $x = 1$, so that in a close range the sharpening loss could be neglected in comparison to BCE.

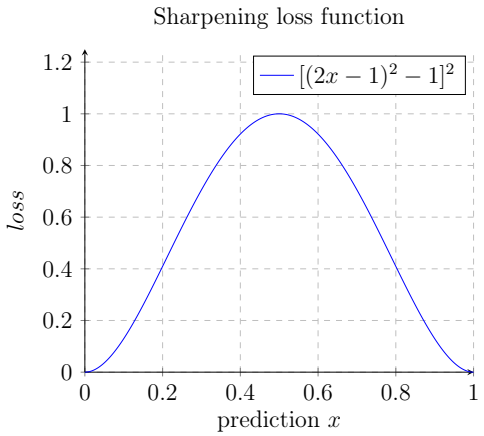


Figure 27: Plotted sharpening loss function. The network tries to minimize this loss so that its output is pushed towards 0 or 1. Near to these two points the gradients decrease so that the classification loss is clearly superior.

All these losses are taken into account for the proposed network architecture. Due to all of them are creating losses on possibly different scales, the weights of all have to be fine tuned.

One goal of the network architecture is tracking. To enable this the framework Deep Tracking [58] as described in section 2.4.1 is used. Due to Deep Tracking requires to mask out the gradients for occluded pixels in the loss calculation it has to be taken into account for the loss components.

Binary cross-entropy is the classification loss of the system and so the basic loss which compares ground truth with prediction. It is necessary to mask out the gradients here because otherwise the occluded pixels would be trained on "free".
The discriminator does not compare ground truth and prediction pixel by pixel. However if the masking would not be used here the discriminator could always distinguish between generation and ground truth by looking at the shapes of the objects. In ground truth there are only lines of the object that can be seen. On the other side the generated frames should ideally contain the whole object what would be punished by the discriminator. So for the GAN framework the masking with the ground truth occlusion map must also be used. When doing this an important point of the GAN get lost: the flexibility of the generator. If the predicted sequence differs from the ground truth the occlusion maps are still the original ones and could lead to vanishing objects or still seeing the full shape. The capability of GANs with Deep Tracking is tested in the experiments in section 4.
The sharpening loss is only based on the prediction image and is still usable for occluded pixels. So the masking is not necessary for that.

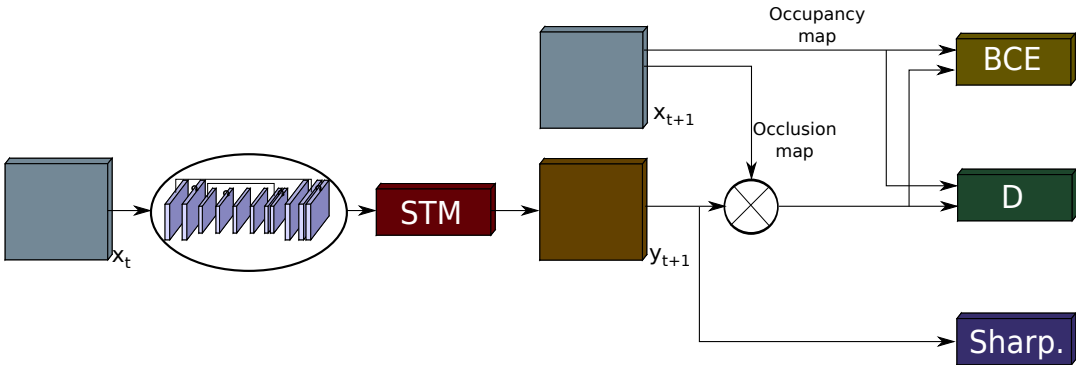


Figure 28: The generator predicts the next frame y_{t+1} based on the previous frame x_t. The loss is calculated by using BCE and a discriminator for which the gradients have to be masked out by using the occlusion map. The sharpening loss solely takes the generated frame. Overall the loss is a weighted sum of all three.

To summarize the loss calculation figure 28 gives an overview of the combination and application of all losses. The input frame x_t is given to the generator which predict the next frame. To compensate the ego vehicle motion an Spatial Transformer Module [36] is applied on it to get the final generated frame y_{t+1}. The prediction is masked out by the ground truth occlusion map from x_{t+1} and provided as input to the Binary cross-

entropy loss and the recurrent discriminator. The sharpening loss does neither need the masking nor the ground truth and so only takes the predicted frame. All of them are weighted and summed up to get the overall loss.

4. Experimental results

To evaluate the network's performance, this section gives an overview of all experiments that were implemented for the presented models. First the MCNet [80] of section 2.4.3 is tested on datasets with standing and moving ego vehicle. Therefore different inputs like motion maps, as described in section 3.1.4, are taken into account. Furthermore the proposed network architecture of section 3.2 analyzed by evaluating different sets of parameters and losses. The section concludes with a comparison and discussion of all models' results.

The evaluation is mostly based on a visual comparison between generated and ground truth images because the loss and metrics are different for every model. In addition for the new network architecture the real images are not available so that a quantitative result is not possible.

4.1. MCNet for environment prediction

Many different network architectures for video prediction can be taken as comparison to the model of section 3.2. However there are some key facts about the dataset what sorts out the best network:

- Compared to video datasets like KTH [69] and UFC-101 [73] contains traffic data only small movements. The network has to attend to these small areas.
- Approaches like [19] are based on translation of pixels. When cars for example turn they change their shape because the sensors detect different sides of it. So a generation of pixels have to be possible.

One recent published network that meets the requirements and counts to the state-of-the-art architecture for video prediction, is the Motion-Content Network [80]. To proof its full capability the model is first trained on an intersection sequence with a standing ego vehicle, and afterwards on a more challenging task with a moving ego vehicle.

4.1.1. Static ego vehicle

A dataset of an intersection with a static ego vehicle was recorded to test whether the MCNet is able detect and predict interactions between objects. If it would already fail on this dataset then the training with a moving ego vehicle would probably perform not better.

The Motion-Content network takes the current image and the difference of the last two frames as input. Since [80] has shown that this architecture performs quite well on RGB images, the recorded data is converted into a similar format. The occupancy map is used as background in that occupied pixels are represented by black, and free space by light gray. It is not a boolean map like in section 3.1.3 because the velocity map

encodes the speed orientation with a color wheel as described in section 3.1.2. Therefore white would not distinguish from a moving pixel as much as it does from light gray. In addition occluded pixels are encoded in dark gray to give the network a hint which pixels are fully visible and which not.

For the training the network gets to see 10 ground truth images and then predicts 10 more. Contrary to the proposed network of section 3.2 there are no generated images for the first part seeing the ground truth. This is done to let the network extract the context of the video by itself and not punishing it for wrong predictions in an early state. However half of the sequence is not used as training data then. The output is fed back as input to predict longer sequences. At this part the difference is calculated on the generated images and not on the ground truth what is done first.

The dataset comprises over 27,000 consequent training frames and 11,000 for testing purpose. The maximum resolution for training is 64x64 pixels limited by the GPU memory of 12GB. At this size a car would be only one pixel wide so that it is not capable. In addition downscaling the image leads to a higher noise ratio in the ground truth and therefore also in the prediction. As compromise every frame is scaled to 256x256 pixels resolution but cropped randomly to 64x64 pixels afterwards. When the network is tested and no gradients have to be calculated the full image can be taken as input. All parameters are summarized in table 1.

Parameter	Description
Input	Velocity map combined with occupancy map
	Difference of last frames
Output	Velocity map combined with occupancy map
Training framework	10 ground truth input, 10 predict
	Output fed back into network for prediction
Training dataset	27,345 frames (45 minutes)
Test dataset	11,527 frames (20 minutes)
Resolution	Training: 64x64 pixels, 20 meters
	Testing: 256x256 pixels, 80.1 meters
Frame rate	10 frames per second

Table 1: Overview of MCNet training parameters for static ego vehicle

Both input and output are RGB images. The ground truth is exactly known by the next frames so that a quantitative evaluation is possible. A common used metric for the comparison is the Mean squared-error (MSE) [24] as defined by equation 2.1.

Nevertheless the MCNet does not train on MSE because [50] has shown that this leads to blurry predictions. This is why another metric should be used that also takes the sharpness of the image into account. One of them is the Peak signal-to-noise ratio (PSNR) that is defined as follows [50, 33]:

$$\mathrm{PSNR}(Y, \hat{Y}) = 10 \log_{10} \frac{\max_{\hat{Y}}^2}{\mathrm{MSE}(Y, \hat{Y})} \tag{4.1}$$

PSNR is the ratio between the maximum possible power of a signal $\max_{\hat{Y}}$ and the noise corruption measured by the mean squared-error between the ground truth Y and prediction $\hat{Y}$. The values are usually expressed in terms of the logarithm decibel scale.

For video prediction the peak signal-to-noise ratio is calculated for every time step and averaged over all test sequences. The result is a curve of the PSNR values along the time dimension as it can be seen in figure 29 for the MCNet. The predictions of the first frames are quite accurate but drop over time. This is a common behavior because the noise increases as the network gets more uncertain for long-term predictions. Compared to the result on the KTH [69] and UCF-101 [73] dataset by [80] the values are quite high and therefore the predictions more accurate. However it is still important to do a visual evaluation to analyze the problems and issues of the network.

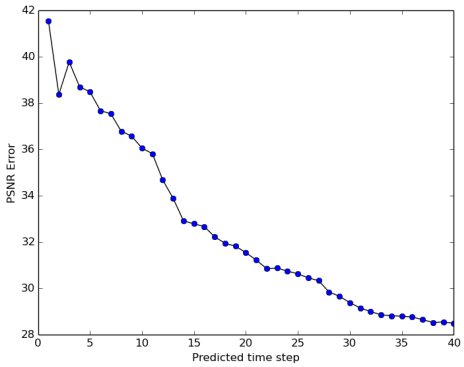


Figure 29: The peak signal-to-noise ratio is calculated for every time step and averaged over all test sequences. The y dimension has logarithm decibel as unit and the x dimension the number of time steps in the future.

Figure 30 shows a sample test sequence of the trained model. The top row consists of the ground truth frames of time steps 1 to 31 in intervals of five. As counter part the bottom row shows the corresponding generated images.
The predictions are quite accurate in the beginning but the noise increases over time. The sequence also shows the two main problems of the model:

- **Objects that become occluded** are partially not visible anymore on the ground truth. So the network has to learn to remove objects that enter occluded areas. However what enters the occlusion has also to come out. This is why the model has to recover these objects again if necessary. Figure 30 shows this problem with a bus at the top. The vehicle enters the occlusion at $t = 6$ and is fully visible at $t = 16$ again. In the generated images the bus is getting smaller and smaller until it is completely removed at $t = 21$. Also the occlusion is not created correctly what highlights this problem.

- **Objects that come out of the occlusion and were not seen before** challenges the network. Though these objects are not predictable the model is punished for not generating them. This leads to random vehicles coming out of occlusions that are hallucinated by the network. Mostly that happens at late time steps like in figure 30 this phenomena begins at $t = 21$ and ends up with two new cars driving in random directions.

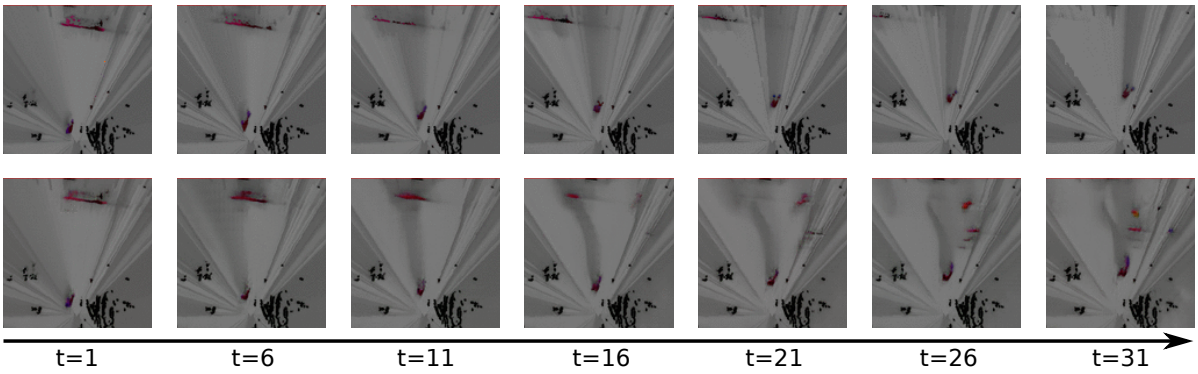


Figure 30: The top row shows the ground truth and the bottom row the generated frames for multiple time steps. This sequence points out the occlusion problem of the network architecture with hallucinated vehicles at $t = 21$ to $t = 31$ and the disappearance of the bus at $t = 11$ and $t = 16$.

This issue could be tackled by Deep Tracking [58]. However tests on this approach does not lead to the expected results. Moreover the occluded area is covered by random and noisy pixel values. Also trainings with a noisy occlusion map that calculates the loss with a probability of 0.5 on an occluded pixel, results in the same occlusion problem as described before. So why is deep tracking not working on RGB images?
An occupancy map is boolean so that an obstacle can be at a certain pixel or not. For occluded space there is always every pixel free and have a fixed value. But in RGB images there is no such default value. Setting occluded pixels to 0 would lead to a black part of the image what encodes occupancy in this dataset. If the pixels are the same as in the original image then it is easier for the network to just copy this part from the input to the output. In addition if the model predicts a different trajectory for an object it gets more punished for predicting the whole shape of the object then just two sides. Overall the occlusion issue can not be solved that easy for a RGB image.

4.1.2. Moving ego vehicle

The application of the prediction systems in real-world requires an ability to also work on a moving vehicle. For this a rather small dataset of 19,500 frames was recorded. The input format of the static ego vehicle dataset can not be used for this one because it does not show the movement of the ego vehicle. For example on a clear street the model can not distinguish static from moving objects without knowing the own speed. This is why the input is extended by using the semantic and horizon map. Sample frames are shown in figure 31.

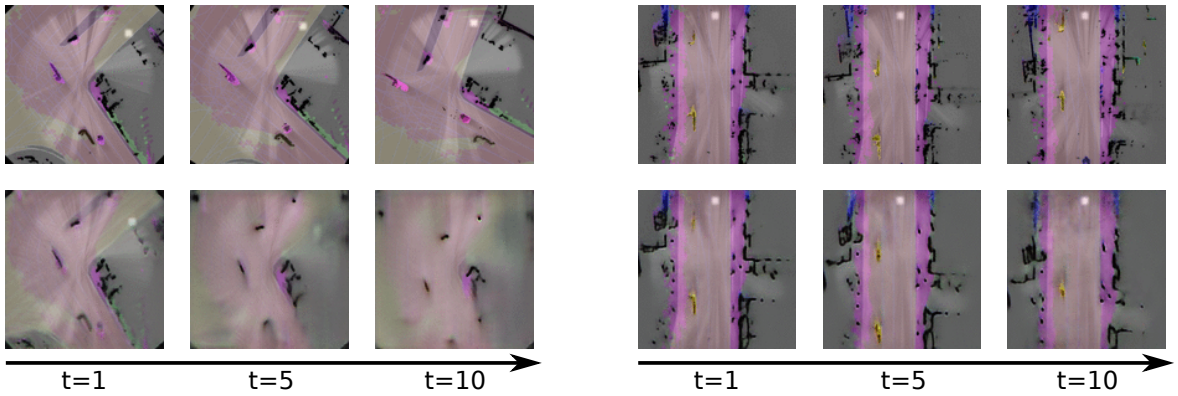
Small cropped resolutions are not capable for moving sequences because new areas come into the field of view and for cropped frames the proportion between new unseen and known pixels is higher than at full resolution. This is why the images are scaled down to 128x128 pixels to compromise on the available GPU memory. Table 2 gives an overview of all training parameters.

Parameter	Description
Input	Combination of velocity, occupancy, horizon and semantic map to RGB
	Difference of last frames
Output	Combination of velocity, occupancy, horizon and semantic map to RGB
Training framework	10 ground truth input, 10 predict
	Output fed back into network for prediction
Training dataset	19,512 frames (33 minutes)
Test dataset	2,400 frames (4 minutes)
Resolution	128x128 pixels, 45.6 meters
Frame rate	10 frames per second

Table 2: Overview of MCNet training parameters for moving ego vehicle

The training is done in the same manner as for the static ego vehicle. Qualitative results are shown in figure 31 while separating moving straight and turning. The number of predicted frames is only 10 for testing because going further would mean that the network has to predict an image of an area that it has not seen at all.

For a straight moving ego vehicle in figure 31b the predictions are quite clear. Both cars on the oncoming lane are translated according to the ego motion and their own speed. Even the one at the top that is not clear to see at $t = 1$ was detected by the network and shifted all the way until $t = 10$. The ground truth contains 2 new car at



(a) Turning ego vehicle (b) Straight moving

Figure 31: (a) When the ego vehicle turns, the network has to rotate the whole image. The top row shows the ground truth and the bottom the generated images. The result is a very blurry output with no usage at all. (b) Straight movements are better to handle for the network because it only has to translate it. Therefore the predictions look much clearer except the part at the top of $t = 10$ where the network has to create a new area.

$t = 10$ that the network could not have known of. Therefore it also does not predict them or any other car. This is an improvement to previous set up where occlusion causes hallucinations of new vehicles.

However turns challenges the network. Figure 31a shows an example of such a scenario. While the prediction at $t = 1$ is slightly noisy the further generated output are not usable at all. Instead of rotating the network shifts the whole image down and blurs it out. Therefore it completely fails on this task. Rotation is much harder to learn because the network has to consider the whole image for that. It has to know where the center in relation to every pixel is, and also which direction. In addition every pixel is transformed differently according to its shift. In contrast translation is the same for every pixel and therefore does not require a huge receptive field. This local transformation can be captured in a few features that are the same for the whole image.

The rotation problem can be solved by using a Spatial Transformer Module [36] as it is used for the network in the next section. Nevertheless the MCNet depends on the difference between frames and using a STM would strongly influence this input. Due to time constraints and a different, more promising approach, this experiment was not implemented and therefore not tested.

4.2. Prediction evaluation on moving vehicle

The following section evaluates the network proposed in section 3.2. Therefore the general experimental setup, including for example dataset and input structure, is described first. Further the architecture is evaluated focusing on different parameter settings for the keep-alive function and the training process. Afterwards the loss components are analyzed and their impact on the results considered.

4.2.1. Experimental setup

For this network the tracking dataset of section 3.1.3 can be used. The input consists of four channels: occupancy, occlusion, road and lanes map. The road and lanes map are extracted out of the horizon map so that they are boolean like occupancy and occlusion map. As additional input the ego vehicle action, including velocity, acceleration and yaw rate, is taken into account. To adjust their range of values to the inner feature maps, the velocity is normalized with 10 m/s as maximum, acceleration as the difference of the normalized velocities and the yaw rate with 10 degrees/s as maximum. The output is only the occupancy map of the next time step.

The training procedure is similar to the MCNet but differs in the fact that also the part with the ground truth inputs is trained. Furthermore this sequence of 10 ground truth and 10 predictions is repeated four times so that the network can better learn to track objects over multiple time steps [58]. For the prediction part the output is fed back into the network as input and the occlusion map is everywhere zero. This represents a fully occluded environment.

The dataset consists of over 80,000 frames for training. This provides a variety of different scenarios. Due to the map is input to the image and the network should not learn false dependencies between a certain map and environment state, the same route is driven multiple times. Consequently the dataset contains some different traffic scenarios for the same positions. In addition the drives were made at different times of the day hence some contain light traffic and some rush hour.

The image resolution is a big constrained for this network because the overall 80 time steps require a huge part of the GPU memory. So the gridmap is set to a size of 45.6 meters encoded in 96x96 pixels. The high resolution is not needed because the occupancy input only consists of boolean and therefore does not get noisy by scaling it down.

In contrast to the tests on the MCNet the frame rate is also considered as flexible parameter. The dataset is recorded with 10 frames per second. This leads to a movement of one or two pixel maximum for every car. Extracting the precise velocity is harder and therefore the predictions much noisier. In addition the noisy increases with the number of predictions. To reduce this amount the frame rate is dropped to 5 frames per second.

The overall experimental setup is summarized in table 3. Different parameter combinations are tested and evaluated in the following paragraphs.

Parameter	Description
Input	Occupancy, occlusion, road and lanes map
	Velocity, acceleration and yaw rate of ego vehicle
Output	Occupancy map
Training framework	10 ground truth input, 10 predict (4 times, alternating)
	Output fed back into network for prediction
	Occlusion map is zero for prediction
Training dataset	80,000 frames (133 minutes)
Test dataset	15,000 frames (25 minutes)
Resolution	96x96 pixels, 45.6 meters
Frame rate	5 frames per second
	10 frames per second

Table 3: Overview of training parameters for proposed prediction network

For the following paragraph the sample results are presented in a certain structure. The top row contains the ground truth frames for the sequence while the predictions are arranged below. Multiple time steps of the same sequence are shown from left to right whereas the network's output is fed back as input. Before the prediction the model has seen 30 ground truth frames to extract the context of the sequence. To give a better comparison between ground truth and prediction the real occlusion map is added to the generated frames. Further the horizon map is provided as background to visualize the environment and context of the sequence.

4.2.2. Architecture evaluation

This subsection deals with different architecture parameters including keep-alive function, frame rate, action conditional predictions and the dense block. Therefore the network was tested on two different setups where every time only one parameter was changed. The results of these two tests are compared and discussed for concluding a final configuration for this concept.

Keep-alive function The keep-alive function is applied between time steps to refresh the network's prediction and remove the noise before feeding it back as input. Therefore different factors a were tested in the range of $[2.5, 10]$ for the function $f(x) = \tanh(a \cdot x)$ and compared to the standard results.
When the keep-alive function is used during training the network notices that the input has changed. The model adjusts its parameter therefore that the output is still the same and contains some uncertainty. In addition the ConvLSTMs remember the last states so that it saves which object was uncertain and hence blurs out these pixel. Applying the keep-alive function only during testing leads to a uncontrolled behavior because the network has never seen such a scenario. The only way around this problem would be by adding a keep-alive function also for the ConvLSTM's hidden states. But these features have a continuous range of values so that it can not be separated like the boolean occupancy map. In addition the hidden states are an internal representation and therefore can not be interpreted easily. This is why the keep-alive function is not further applied for this model.

Action conditional predictions The input of the ego vehicle's action is not only used for simulation purpose but it also helps the network to predict interactions with the other objects. Without this extra input the model would not know about the velocity and yaw rate of the ego vehicle because it is compensated by the Spatial Transformer Modules [36]. Therefore the prediction system is not able to infer any relations between the ego and other vehicle's motion. For example shows figure 32 where the ego vehicle slows down and finally stops at an intersection. Vehicles on the same lane would most likely reduce their speed too but the network predicts a continuous motion. Furthermore the car behind the ego vehicle crashes into it in the model's predictions. Hence using action-conditioned input and the ego vehicle in the occupancy map is indispensable.

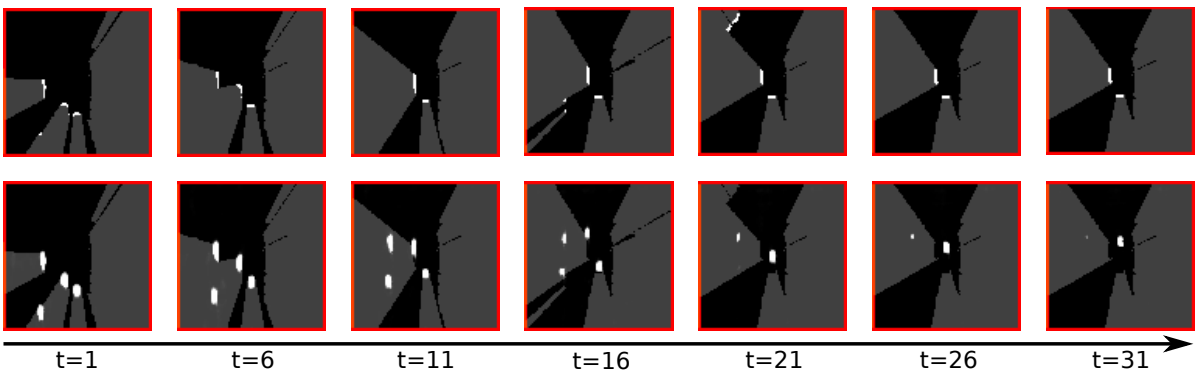


Figure 32: The ground truth at the top shows a stopping sequence. However the network predicts that the other vehicles would keep going and even crash into the ego vehicle. Due to this phenomena was recognized in an early state of this work the ego vehicle and the horizon map were not visualized yet.

Frame rate As described in the experimental setup two different frame rates were tested for the proposed network. The lower the frame rate is, the higher gets the uncertainty of an object in the next frame. However it reduces the number of images that have to be predicted for a certain time period so that the impact of the noise gets smaller.
Tests on 5 and 10 frames per second have shown a similar behavior. Since the ground truth input is very noisy too, the network can better distinguish between noise and real objects for lower frame rates. In contrast the model with 10 frames per second seems to capture the object's velocity more precise.
However the model with 5 frames per second has one crucial advantage: Interactions most likely proceed over a longer time period instead of hundred milliseconds. This is why fewer frames compress this interactions and the network has not to learn huge long-time dependencies which are harder to learn as mentioned in section 2.3.2.
Figure 32 contains an example for 10 frames per second training. Therefore the objects are getting smaller and smaller over time steps so that most of them vanished at $t = 31$. From the time perspective out this would mean that the system can not predict longer than 3 seconds. As comparison figure 33 shows the same sequence for a network trained with 5 frames per second and action-conditional input. Here the objects are more stable and even if it looses the upper left vehicle the car behind it is correctly predicted in this alternative environment. The vehicle does not stop in the middle of the road but more at the intersection next to the other cars. In addition through the action-conditional input the network is able to predict a similar stopping behavior for the surrounding cars.

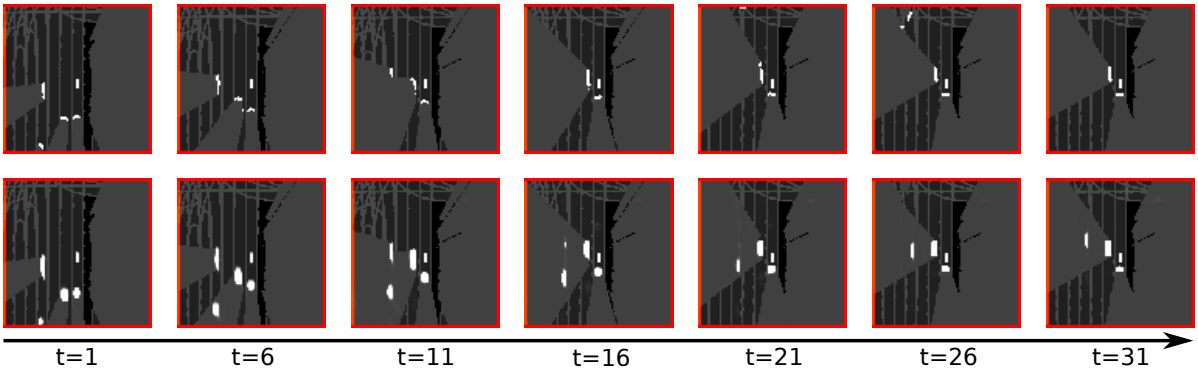


Figure 33: The ground truth sequence is the same as in figure 32 but this time the prediction network was trained on 5 frames per second and an action-conditional input. The predictions look clearer and the possible prediction time is longer because $t = 31$ is here 6.2 seconds in the future.

Dense block In this paragraph the usage of the dense block is evaluated. As an alternative the block is replaced by two dilated convolutions that results in a receptive

field of about 30x30. The network is therefore smaller and in execution time 5% faster.
Tests without the dense net block show only local connectivity. Every output pixel depends on a small area of the input and therefore the whole context can not be considered for predictions. The dense block provides a receptive field of over 120x120 that is even bigger than the image resolution. However neurons on the edges can only use half of field because the other half just looks at the zero paddings of the convolutions. The dense block shows multiple advantages. On the one hand it speeds up the training because the gradients can flow back better over the skip connections. Furthermore the network gets a context understanding so that its prediction depends on the map environment (intersection, straight road, a.s.o.). A negative effect of this can be seen in figure 34. When the network is trained on a small dataset in which the traffic scenarios for a certain map pattern are always the same, then the model overfits on this situation. Figure 34 shows the results of a network that has seen for every intersection two cars left of ego vehicle. The test sequence is a intersection but now without cars next to it. However the network still predicts them although they are not there.

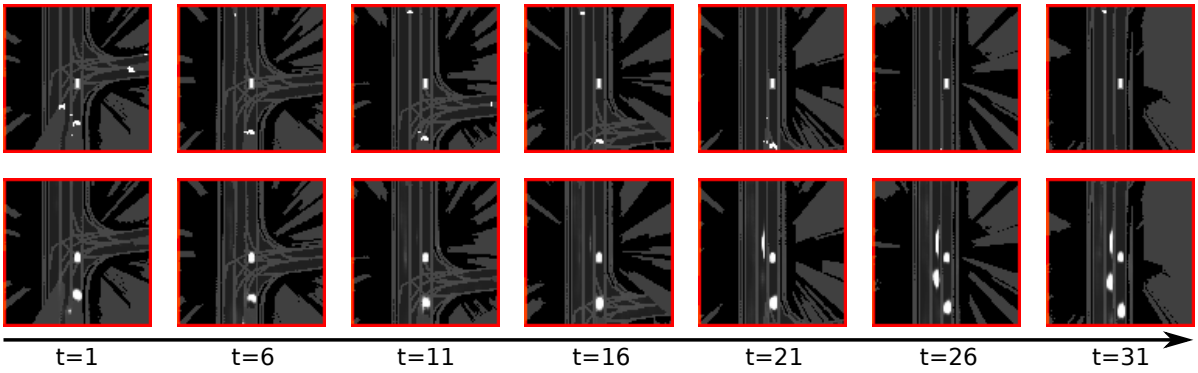


Figure 34: The network was only trained on intersection scenarios where two cars are left to the ego vehicle. Therefore the prediction is biased and can not generalize to new intersection scenarios as the one in this figure. The result is a hallucination of two cars on the left because they have been there in every other training sample.

4.2.3. Loss evaluation

Besides the architecture setup, the loss calculation of section 3.2.3 was another main point of the new network. To evaluate the loss structure different combinations and weight factors were tested and analyzed. This section summarizes the results for every loss separately and compares combinations of different components.

Binary cross-entropy As a standard image for classification task, binary cross-entropy (BCE) constitutes the fundamental loss on which the network optimizes its parameter. However the default version of BCE punishes zeros and ones equally although the dataset is strongly biased on zeros. In addition it is more critical for the prediction task to estimate a pixel as free when it was occupied, than the other way round. Therefore the loss function was extended by two parameters: a weight factor for the one's loss and a stretch measure for the zero's loss to prevent a loss of infinity. Nevertheless the values of both factors is not fixed and have to be adjusted. The default parameter setting for the standard binary cross-entropy is when both factors are equals one.

First the weight factor for the one's loss is evaluated. Small parameter values under 2 seem not have a noticeable impact on the networks performance. However if a too high factor is chosen the network starts to randomly create new objects or just spread the object's shape instead of translating the pixel. Thus the range of values is approximately limited between 2 and 3. The exact number depends on the second parameter that is a stretch factor for the zero's logarithm function.
The parameter has to be smaller than one to stretch the logarithm. With a factor of 0.5 the maximum loss would only be 0.3 for predicting a free pixel as occupied. In contrast the same loss gets the network for estimating a occupied pixel with 0.5. So small factors lead to fully occupied prediction environments and therefore is not usable. The boundaries for this parameter are around 0.9 and 1. As a compromise between both parameters a weight factor of 2 and stretching of 0.9 worked the best for all experiments. Even so the logarithm functions, as defined by the equation 3.3, head to infinity what can not be represented and calculated precisely in most programming languages. So a small offset of $\epsilon = 10^{-10}$ is added or rather subtracted to both logarithms for approximation.

GAN Discriminator GANs [25] got impressive results for image generation and were therefore suggested as another loss component for this task. A discriminator network takes multiple consequent frames as input and decides whether these images are generated our ground truth. The aim of the generating network is to fool the discriminator while the other one is trained to distinguish predicted and ground truth frames as good as possible. Therefore the generator has to simulate real data what implies different challenges for the occupancy map:

- The ground truth is recorded sensor data so that noise is indispensable. So to trick the discriminator, the generator has also to learn and adapt noise into its own predictions. But this is not what the goal of the prediction system is. Furthermore the binary cross-entropy loss would work against this noise injection because the ground truth noise follows a certain pattern.
- Due to the network is trained on the Deep Tracking framework [58] the output

contains the full shape of the vehicle. This is a clear criterion for the discriminator to separate predictions from real data. The only way around that is using the ground truth occlusion map for masking out the generated frames. Therefore if the generator predicts a slightly different movement than in the ground truth the discriminator easily detects that and forces the prediction system to overfit on every scenario.

This is why a combined loss of binary cross-entropy and GAN discriminator does not perform better than a simple BCE. During training the discriminator always is superior to the generator and could only be fooled with random predictions.

Sharpening loss As the last component the sharpening loss is evaluated. The function punishes the network for making uncertain predictions around 0.5 and pushes the values to 0 or 1. At the beginning of the training the network predicts random values. In the first iteration it is strongly pushed to predict zeros by the binary cross-entropy because even with the weighted function the loss is lower for predicting no occupancy than a full occupied map. Afterwards the network starts to increase the prediction values step by step until it stabilizes in a lower minimum. In this period the sharpening loss would work against the binary-cross entropy because it is more expensive to predict a probably occupied pixel with 0.5. This is why the weight factor of the sharpening loss is set to zero for the first 2,000 iterations and then raised to one. Figure 35 compares the network's result trained with or without the sharpening loss. Using only binary cross-entropy leads to blurry predictions as the bottom row shows. In addition the objects vanish quickly and a false point cloud is visible in the occlusion at $t = 6$ to $t = 16$. In contrast the results from the network with sharpening loss are clear and more distinct.

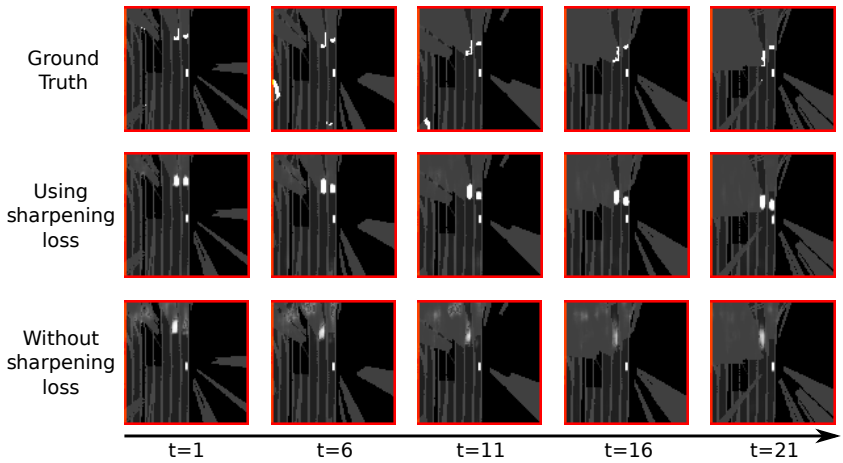


Figure 35: The bottom row shows the results for a network trained solely on BCE while for the center row the sharpening loss was used too. Not only the predictions are getting much more distinct but also the objects can be extracted better.

4.3. Comparison and Discussion

The following paragraph compares the results of the MCNet [80] with the ones of the new proposed network. First the systems are evaluated on all challenges listed in section 3.2 and afterwards discussed for further improvements.

Noisy data The ground truth, that is recorded by real-world sensors, most likely contains noise. Therefore the network has to learn a robust representation and must distinguish between correct object detections and random noise.
The MCNet's predictions become instable if noisy points are detected near the occlusion. Then the network sometimes hallucinates new objects entering the field of view. The proposed network is less sensitive to noise because it is trained with Deep Tracking. Therefore it can better decide whether a real object can come out of the occlusion or if it is just noise. However if the network only sees an object on the last one or two frames, it most likely looses this object rather than predicting its trajectory other multiple time steps. So it is not completely resistant against noise and can not predict short-time seen objects.

Biased dataset Most of the grid map consists of free space where no motion is detected. However the prediction system has to focus on moving objects because they can affect and interact with the ego vehicle.
The architecture of the MCNet splits motion and content into two different encoders. So network can separately concentrate on moving objects and predict their interaction among each other. To incorporate the overall context of the traffic environment the output of the content and motion encoder are concatenated together for the final prediction. Figure 30 shows an example of the MCNet where only a subpart of the image contains movements. However the network does not change any pixel of the static environment and fully focuses on the moving objects.
For the proposed network of section 3.2 the binary cross-entropy is weighted so that it gets a higher loss for predicting an occupied pixel wrong. In addition the logarithm function for the zeros is stretched to prevent a loss of infinity.

Long-distance interactions Objects can interact with each other over a huge distance. Therefore the whole image has to be taken into account to predict the next step. However the Motion-Content Network does not have a significant higher receptive field than other networks. The predictions are still based on local observations and not really set into the overall context. In contrast to that the proposed network contains the dilated dense block which increases the field of view enormously. As a result the overall traffic scenario like a four way intersection is recognized and considered for the prediction. Nevertheless this leads to a scenario overfitting as shown in figure 34 if not enough traffic variations are in the training dataset.

Occlusion Occluded objects are one of the big challenges for the network. Especially the MCNet on static ego vehicle fails on tracking and predicting objects through occlusion. In figure 30 for example the bus at the top of the image is getting occluded but the network looses it and does not regenerate it behind the occlusion. Furthermore new vehicles are randomly predicted to come out of the occlusion.
The proposed network is trained with the Deep Tracking [58] framework. Therefore it learns that objects can get occluded but still keeps them until they become visible again. One example of that is shown in figure 33.

Relative motion The prediction system has to able to deal with ego vehicle motion. The problem is that whole surrounding relatively moves regarding to the ego vehicle on the gridmap.
For the MCNet the motion encoder can recognize the translation of the image. Although the results on the straight movements are quite well, the network fails to predict the environment for a turning scenario. In contrast the Spatial Transformer Module (STM) [36] relieves the network from this challenge. The task is broken down to predict the environment for a standing ego vehicle. This output is transformed by the STM to compensate the ego vehicle motion. Turns become much more easier for the network, however the rotation does not fit perfectly to the ground truth because the recorded velocity and yaw rate only gives an approximation of the real motion.

Unseen objects Dealing with unseen objects remains an open task. Even the network can not predict them with an 100% certainty, many situations infer that a new vehicle will most likely enter the field of view. One example for this problem at an intersection is shown in appendix A. Two cars want to turn but stopped for a longer time. Therefore they will probably wait on a car to pass coming from the left. However the MCNet predicts that the car on the top starts to turn and therefore does not consider the new vehicle. Moreover after two seconds the network begins to generate new objects coming out of the left occlusion. This has no apparent reason but the problem is more that the network was trained on unseen objects. So it also guesses that new vehicles can come out there although it is not the case. The same problem can be observed for the other network. So this stays an open challenge that have to be solved.

Real time To apply a prediction system in the car the execution time has to be as small as possible. The minimum time would be 200ms for 5fps or 100ms for 10fps because the prediction should be updated with every new frame. The MCNet can predict one second in this period of time. In comparison the proposed network is able to predict 5 seconds in 200ms execution time. However if multiple trajectories should be tested then it has to be done on multiple GPUs in parallel. This requires a stronger and much more expensive computer although control devices in the car

can not provide that.

Combination of all challenges The final challenge is to combine all these problems and find one fitting loss function for this complex network. The MCNet an ℓ_2 loss combined with the GAN framework to generate realistic predictions. However many challenges are solved on the way or not taken into account. This is one reason for the bad performance on occluded objects.
The loss calculation for the new network is much more complicated and specialized. On the one hand Deep Tracking is used by masking out the gradients in the occluded area. In addition the sharpening loss and discriminator helps the network to learn a sufficient object representation. All challenges are solved by different modules or subcomponents and only the combination of all leads to a good prediction.

Based on the challenges the new proposed network outperforms the state-of-the-art video prediction models like the MCNet. To conclude the evaluation further results and problems are discussed.

Appendix B contains multiple sample results that show different aspects of the network's performance. For example shows figure 40 the map understanding of the network. The car on the right lane was going straight in the last few frames that the network has seen as input. However its lane is turning to the right. Therefore the network recognizes this orientation change and predicts a certain turn to the right.
In this scenario new cars enter the field of view from the bottom but the network does not predict these. Nevertheless a human could also not guess if new vehicles will follow or not. Furthermore the car on the bottom left is only detected on the last two or three frames and so the network could not distinguish it from noise. This is a critical issue because if such vehicles enter with a high speed it will affect the ego vehicle within the next second and has to be predicted as soon as possible.

Multimodal situations are another problem for the network as shown in figure 41. The vehicle at the top has multiple possible trajectories which it can take. For example it can further stop, driving straight or turn left. The network has to commit itself on one of these possibilities. However the loss is higher if the network predicts the wrong trajectory then just loosing this object. Therefore the network starts to predict the left turn in figure 41 but the vehicle vanishes are a few frame. This problem remains as an open challenge for future work that is discussed further in the outlook of section 5.

5. Outlook

The proposed network architecture turns out to be a good approach for dealing with the task of Deep Prediction Learning. However some open challenges still remain that have to be solved before such a system can be applied for autonomous driving. Furthermore the model has to pass several tests and its reliability must be proven. For neural networks there is no simple unit test that verifies its performance and correctness [24, 53]. Therefore resolving all known and open challenges are even more important for these systems.

One of the open challenges is increasing the gridmap size. Right now the network is running at a resolution of only 96x96 pixels and a range of 45.6 meters but a much wider field of view is needed for predicting intersection scenarios. However the size for training is limited by the maximum GPU memory of 12GB. Besides the resolution the batch size and the number of future frames, on which the network is trained at every iteration, influences the memory usage and can be scaled down. Nevertheless a higher batch size guarantees a better generalization, and many future frames are needed to recognize long-time dependencies. So raising the gridmap size requires a decreasing of one of the other parameters but this leads to different disadvantages. For inference time using a higher resolution is not an issue because no gradients have to be stored over multiple time steps. One possible way to go is to still train the network on a low resolution but test it on a bigger size. Even so the model will probably not take the wider context into account because it has learned from a smaller size. Therefore this challenge is not solved yet.

The occupancy map of the gridmap relies on raw lidar points. Other prediction approaches like [3, 45, 60] are based on an object representation like figure 36 and predict a trajectory for every single object. For this the model focuses on interactions and does not have to tackle noise or object recognition. Still this kind of data is not every time available like it is the case for this thesis. However to validate the interaction learning of the proposed network a comparison with an object based system is necessary. Alternatively the semantics and velocity map can be added as an additional input to enrich the environment information. Nevertheless these input channels should be blacked out while predicting in order to reduce the complexity of the output.

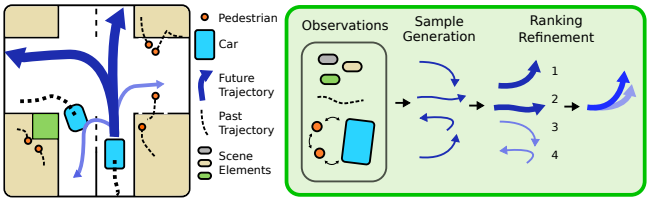


Figure 36: The object representation used by the DESIRE system [45] contains cars and pedestrians. Based on their past movements multiple trajectories are sampled and ranked to propose the final prediction.

Another big problem is the prediction of unseen objects. When the ground truth consists of several future frames the network is trained on predicting objects that thereby enter the field of view. This leads to systems that hallucinate random new vehicles like in figure 30. Motion maps can help to limit the probability of entering objects. It summarizes all common driver behaviors and implies traffic rules. In addition the pedestrian map in figure 21c contains crosswalks of intersections and therefore points out possible positions where a pedestrian can enter the field of view. So this structure gives the network a hint where new objects can appear and how they most likely behave. However there is never a 100% probability of an unseen object entering the field of view hence the prediction just remains as a guess.

The biggest challenge of constructing a network for the prediction task is the loss calculation. As described in section 3.2.3 the current function is a combination of weighted binary cross-entropy [24], a GAN discriminator [25] and the sharpening loss. However it turns out that the discriminator is not suitable because it forces the prediction network to generate noise. In addition using the occlusion mask for the deep tracking framework takes the flexibility of the generator away. Still GANs show impressive results for image generation so that more research on a better loss involvement would be worth it. Another issue is the distribution of predicted and ground truth occupancy pixels as shown in figure 37.

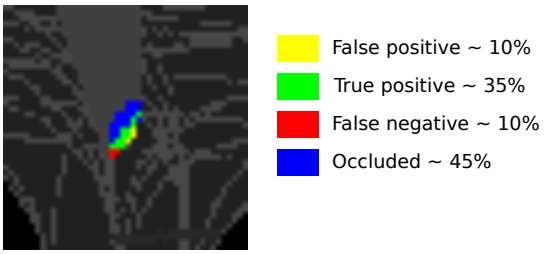


Figure 37: For each frame the pixel loss is calculated on visible parts that are categorized in correct occupied (true positives), missing (false negative) and false occupied (false positive) predictions. Averaged over all frames about 45% of all occupied pixels are occluded and not included in the loss.

The loss is only calculated on visible pixels and therefore categorized in true positive, false positive and false negative. However more than 45% of the predicted pixels are occluded. This only applies if the network predicts the object at the correct position. If the estimation is a little bit off the occluded pixels get visible and are included in the loss as false positive. This happens often for long sequences because an object can randomly accelerate or slow down. The loss contains then all object's pixels as false positive and the ground truth as false negative, but the false positives are about twice as much. If a object has too many possible trajectories the network vanishes this object because it removes the huge amount of false positives. This phenomena is shown in figure 41. To solve this problem an object loss has to be introduced that punishes the

network for loosing pixels. Yet it is not easy to propose a function for this problem so that it remains as an open challenge. Corresponding to that a measurement metric has to be hypothesized to compare the performance of different networks. For this a network predicting a wrong trajectory should still be better than a network that looses the whole object.

The last challenge that should be mentioned here is the execution time. The current system needs about 150-200ms for predicting 5 seconds into the future. These frames depend on a single ego vehicle trajectory that is a input to the network. However testing only one trajectory is not sufficient because the prediction system should support the selection procedure of the best driving path. Therefore the predictions of multiple trajectories are compared and evaluated. This can be performed in parallel but requires several high-end GPUs and is not feasible. In addition the trajectory must be quickly adjusted if the prediction detects a possible crash. So the execution time has to be reduced. Due to the input and ground truth output is binary, the whole network could be approximated by binarising all weights and filters as shown in figure 38. The XNOR network [62] shows similar results with this architecture compared to standard models like the AlexNet [40] but is up to 58 times faster and uses 32 times less memory. During training the gradients are calculated and applied on the exact floating point values but for the forward pass all parameters are approximated to -1 or 1 (see figure 38). For inference time the network solely performs on the binary values and is therefore much faster. This structure can be used for any network architecture and could also help for the proposed prediction model.

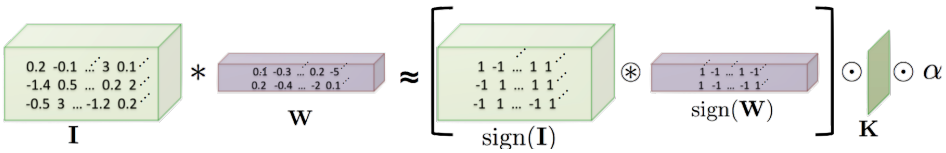


Figure 38: The XNOR network [62] approximates every convolution with binary inputs and weights. To still factorize different filters, **K** provides a weight parameter for every sub-tensor of the input and α for the weights.

Overall there are many open challenges that have to be solved in the future. Therefore this thesis and the proposed network just represents the first step of a long research project. Nevertheless the current results look promising and proves the capability of machine learning for the task of predicting and interpreting interactions between multiple traffic participants.

6. Conclusion

To determine a safe and comfortable trajectory for an autonomous vehicle, the future actions of all surrounding traffic participants have to be predicted. Learning systems like artificial neural networks consider the human as role model and outperform classical methods for the task of environment prediction because of the multitude of different traffic scenarios and driver behaviors. However first approaches like [45, 60] are based on object representations that provide an summary of all traffic participants and their position. This data is expensive to generate and therefore not every time available. Simpler structures like an gridmap, containing occupied and free points of the environment, already provide a data model from which a human is able to predict the next time steps. Hence this thesis deals with the prediction of multiple objects in a traffic environment based on only slightly preprocessed sensor data like occupancy gridmaps.

Multiple challenges arise with this task so that a new network architecture had to be developed. The model structure is inspired by [19] and consists of a encoder-decoder architecture [11] with four convolutional LSTM [70] to remember information over many time steps. Besides the occupancy, the network takes an occlusion gridmap, determining which points are directly visible for the sensors and which not, and the road structure as input. In addition the model considers the speed, acceleration and yaw rate of the ego vehicle to predict trajectory-conditioned interactions. A combination of dilated convolutions [84] with a dense block [32] is used to rapidly increase the receptive field and detect interactions over long ranges.
The output of the network only consists of the next occupancy map. To predict more than one future time step, the generated frame is fed back as input with a blacked out occlusion and translated road map. Thereby the noise is reduced by using a keep-alive function to refresh the occupancy prediction. Due to the ego vehicle always is centered and so every over object's motion is relative to it, a Spatial Transformer Module [36] transforms the output regarding to the ego vehicle speed. This reduces the task towards predicting all traffic participants from a static perspective.
The loss on which the network is trained, combines three components. One is a weighted binary cross-entropy [24] to balance the occupied and free space. In addition a GAN discriminator [25] promotes the multimodality of the predicted output distribution while the sharpening loss further punishes the network for uncertain predictions. Because objects can become occluded but still have to be considered for prediction, the gradients are masked out for occluded parts by using the Deep Tracking framework [58, 16].

For experiments and tests the architecture was trained on eight driving sequences consisting of 10,000 frames each. The network outperforms state-of-the-art models for video prediction like the Motion-Content Network [80] and shows an impressive understanding of the traffic environment and interactions. However many open challenges like the prediction of unseen objects, mutlimodal trajectories and faster execution time, remain that have to be solved before the system can be applied for autonomous driving.

References

[1] Abbeel, P. and Ng, A.Y.: Apprenticeship learning via inverse reinforcement learning. In: Proceedings of the twenty-first international conference on Machine learning, p. 1. ACM, 2004.

[2] Aghdam, H.H. and Heravi, E.J.: Guide to Convolutional Neural Networks: A Practical Application to Traffic-Sign Detection and Classification. Springer International Publishing, Cham (Switzerland), first edition, 2017.

[3] Alahi, A. *et al.*: Social lstm: Human trajectory prediction in crowded spaces. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 961–971, 2016.

[4] Arjovsky, M. and Bottou, L.: Towards principled methods for training generative adversarial networks. *Computing Research Repository (CoRR)*, volume abs/1701.04862, 2017.

[5] Arjovsky, M./ Chintala, S. and Bottou, L.: Wasserstein GAN. *Computing Research Repository (CoRR)*, volume abs/1701.07875, 2017.

[6] Arjovsky, M./ Chintala, S. and Bottou, L.: Wasserstein generative adversarial networks. In: Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017, pp. 214–223, 2017.

[7] Ba, J.L./ Kiros, J.R. and Hinton, G.E.: Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[8] Bengio, Y./ Simard, P. and Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.*, volume 5, no. 2, pp. 157–166, March 1994.

[9] Bojarski, M. *et al.*: End to end learning for self-driving cars. *Computing Research Repository (CoRR)*, volume abs/1604.07316, 2016.

[10] Chen, X. *et al.*: Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In: Advances in Neural Information Processing Systems 29, pp. 2172–2180. Curran Associates, Inc., 2016.

[11] Cho, K. *et al.*: On the properties of neural machine translation: Encoder-decoder approaches. *Computing Research Repository (CoRR)*, volume abs/1409.1259, 2014.

[12] Cordts, M. *et al.*: The cityscapes dataset for semantic urban scene understanding. In: Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[13] da Silva, I.N. *et al.*: Artificial Neural Networks: A Practical Course. Springer International Publishing, Switzerland, first edition, 2017.

[14] DeepDrive: Guiding the next generation of research, 2017. URL: https://deepdrive.berkeley.edu/, last viewed on 10 August 2017.

[15] Denton, E.L. *et al.*: Deep generative image models using a laplacian pyramid of adversarial networks. *Computing Research Repository (CoRR)*, volume abs/1506.05751, 2015.

[16] Dequaire, J. *et al.*: Deep tracking on the move: Learning to track the world from a moving vehicle using recurrent neural networks. *Computing Research Repository (CoRR)*, volume abs/1609.09365, 2016.

[17] drive.ai: Building the brain of self-driving vehicles, 2017. URL: https://www.drive.ai/, last viewed on 10 August 2017.

[18] Everingham, M. *et al.*: The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, volume 88, no. 2, pp. 303–338, June 2010.

[19] Finn, C./ Goodfellow, I.J. and Levine, S.: Unsupervised learning for physical interaction through video prediction. *Computing Research Repository (CoRR)*, volume abs/1605.07157, 2016.

[20] Finn, C. and Levine, S.: Deep visual foresight for planning robot motion. *Computing Research Repository (CoRR)*, volume abs/1610.00696, 2016.

[21] Franke, U. *et al.*: Making bertha see. In: Proceedings of the IEEE International Conference on Computer Vision Workshops, pp. 214–221, 2013.

[22] Gentle, J.: Matrix Algebra: Theory, Computations, and Applications in Statistics. Springer Texts in Statistics. Springer New York, 2007.

[23] Gers, F.A. and Schmidhuber, J.: Recurrent nets that time and count. In: IJCNN (3), pp. 189–194, 2000.

[24] Goodfellow, I./ Bengio, Y. and Courville, A.: Deep Learning. MIT Press, Cambridge, Massachusetts, first edition, 2016.

[25] Goodfellow, I. *et al.*: Generative adversarial nets. In: Z. Ghahramani/ M. Welling/ C. Cortes/ N.D. Lawrence and K.Q. Weinberger, editors, Advances in Neural Information Processing Systems 27, pp. 2672–2680. Curran Associates, Inc., 2014.

[26] Goodfellow, I.J.: NIPS 2016 tutorial: Generative adversarial networks. *Computing Research Repository (CoRR)*, volume abs/1701.00160, 2017.

[27] Gulrajani, I. *et al.*: Improved training of wasserstein gans. *Computing Research Repository (CoRR)*, volume abs/1704.00028, 2017.

[28] He, K. *et al.*: Deep residual learning for image recognition. *Computing Research Repository (CoRR)*, volume abs/1512.03385, 2015.

[29] Hochreiter, S.: Untersuchungen zu dynamischen neuronalen netzen. *Master's thesis, Institut fuer Informatik, Technische Universitaet, Muenchen*, 1991.

[30] Hochreiter, S. and Schmidhuber, J.: Long short-term memory. *Neural Comput.*, volume 9, no. 8, pp. 1735–1780, November 1997.

[31] Holroyd, C.B. and Coles, M.G.: The neural basis of human error processing: reinforcement learning, dopamine, and the error-related negativity. *Psychological review*, volume 109, no. 4, p. 679, 2002.

[32] Huang, G. *et al.*: Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.

[33] Huynh-Thu, Q. and Ghanbari, M.: Scope of validity of psnr in image/video quality assessment. *Electronics letters*, volume 44, no. 13, pp. 800–801, 2008.

[34] Ioffe, S. and Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: F. Bach and D. Blei, editors, Proceedings of the 32nd International Conference on Machine Learning, *Proceedings of Machine Learning Research*, volume 37, pp. 448–456. PMLR, Lille, France, 07–09 Jul 2015.

[35] Isola, P. *et al.*: Image-to-image translation with conditional adversarial networks. *Computing Research Repository (CoRR)*, volume abs/1611.07004, 2016.

[36] Jaderberg, M. *et al.*: Spatial transformer networks. *Computing Research Repository (CoRR)*, volume abs/1506.02025, 2015.

[37] Kaelbling, L.P./ Littman, M.L. and Moore, A.W.: Reinforcement learning: A survey. *Journal of artificial intelligence research*, volume 4, pp. 237–285, 1996.

[38] Karpathy, A.: Cs231n convolutional neural networks for visual recognition, 2017. URL: http://cs231n.github.io/convolutional-networks/, last viewed on 8 August 2017.

[39] Klambauer, G. *et al.*: Self-normalizing neural networks. *Computing Research Repository (CoRR)*, volume abs/1706.02515, 2017.

[40] Krizhevsky, A./ Sutskever, I. and Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems, pp. 1097–1105, 2012.

[41] Kuefler, A. *et al.*: Imitating driver behavior with generative adversarial networks. *arXiv preprint arXiv:1701.06699*, 2017.

[42] LeCun, Y./ Bengio, Y. and Hinton, G.: Deep learning. *Nature*, volume 521, no. 7553, pp. 436–444, 2015.

[43] LeCun, Y. *et al.*: Backpropagation applied to handwritten zip code recognition. *Neural computation*, volume 1, no. 4, pp. 541–551, 1989.

[44] LeCun, Y. *et al.*: Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, volume 86, no. 11, pp. 2278–2324, 1998.

[45] Lee, N. *et al.*: Desire: Distant future prediction in dynamic scenes with interacting agents. *arXiv preprint arXiv:1704.04394*, 2017.

[46] Lin, T. *et al.*: Microsoft COCO: common objects in context. *Computing Research Repository (CoRR)*, volume abs/1405.0312, 2014.

[47] Liu, W. *et al.*: Ssd: Single shot multibox detector. In: European conference on computer vision, pp. 21–37. Springer, 2016.

[48] Luo, W. *et al.*: Understanding the effective receptive field in deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 4898–4906, 2016.

[49] Macadam, C.C.: Understanding and modeling the human driver. *Vehicle System Dynamics*, volume 40, no. 1-3, pp. 101–134, 2003.

[50] Mathieu, M./ Couprie, C. and LeCun, Y.: Deep multi-scale video prediction beyond mean square error. *Computing Research Repository (CoRR)*, volume abs/1511.05440, 2015.

[51] Mitchell, T.M.: Machine Learning. McGraw-Hill, Inc., New York, NY, USA, first edition, 1997.

[52] Mnih, V. *et al.*: Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[53] Morton, J./ Wheeler, T.A. and Kochenderfer, M.J.: Optimal Testing of Self-Driving Cars. *ArXiv e-prints*, July 2017. 1707.08234.

[54] Nguyen-Tuong, D. and Peters, J.: Model learning for robot control: a survey. *Cognitive processing*, volume 12, no. 4, pp. 319–340, 2011.

[55] Nielsen, M.: Neural networks and deep learning.(2016). *Saatavilla http://neuralnetworksanddeeplearning. com/index. html, viitattu*, volume 18, 2016.

[56] Noh, H./ Hong, S. and Han, B.: Learning deconvolution network for semantic segmentation. *Computing Research Repository (CoRR)*, volume abs/1505.04366, 2015.

[57] Olah, C.: Understanding lstm networks, August 2015. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/, last viewed on 28 August 2017.

[58] Ondruska, P. and Posner, I.: Deep tracking: Seeing beyond seeing using recurrent neural networks. *Computing Research Repository (CoRR)*, volume abs/1602.00991, 2016.

[59] Ondruska, P. *et al.*: End-to-end tracking and semantic segmentation using recurrent neural networks. *Computing Research Repository (CoRR)*, volume abs/1604.05091, 2016.

[60] Ortiz, M.G.: Prediction of driver behavior. *Theses for the degree of Doktor-Ingenieur (Dr.-Ing.), Technische Fakultaet, Universitaet Bielefeld*, 2014.

[61] Palazzi, A. *et al.*: Learning to map vehicles into bird's eye view. *Computing Research Repository (CoRR)*, volume abs/1706.08442, 2017.

[62] Rastegari, M. *et al.*: Xnor-net: Imagenet classification using binary convolutional neural networks. In: European Conference on Computer Vision, pp. 525–542. Springer, 2016.

[63] Redmon, J. and Farhadi, A.: Yolo9000: Better, faster, stronger. *Computing Research Repository (CoRR)*, volume abs/1612.08242, 2016.

[64] Redmon, J. *et al.*: You only look once: Unified, real-time object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 779–788, 2016.

[65] Rumelhart, D.E. *et al.*: Learning representations by back-propagating errors. *Cognitive modeling*, volume 5, no. 3, p. 1, 1988.

[66] Russakovsky, O. *et al.*: ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, volume 115, no. 3, pp. 211–252, 2015.

[67] Sak, H./ Senior, A. and Beaufays, F.: Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *arXiv preprint arXiv:1402.1128*, 2014.

[68] Schlechtriemen, J./ Wabersich, K.P. and Kuhnert, K.D.: Wiggling through complex traffic: Planning trajectories constrained by predictions. In: Intelligent Vehicles Symposium (IV), 2016 IEEE, pp. 1293–1300. IEEE, 2016.

[69] Schuldt, C./ Laptev, I. and Caputo, B.: Recognizing human actions: a local svm approach. In: Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on, volume 3, pp. 32–36. IEEE, 2004.

[70] Shi, X. *et al.*: Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *Computing Research Repository (CoRR)*, volume abs/1506.04214, 2015.

[71] Silver, D. *et al.*: Mastering the game of Go with deep neural networks and tree search. *Nature*, volume 529, no. 7587, pp. 484–489, January 2016.

[72] Singh, S.P. *et al.*: Robust reinforcement learning in motion planning. In: Advances in neural information processing systems, pp. 655–662, 1994.

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

[74] Soutner, D. and Müller, L.: Application of LSTM Neural Networks in Language Modelling, pp. 105–112. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[75] Sundermeyer, M./ Schlüter, R. and Ney, H.: Lstm neural networks for language modeling. In: Thirteenth Annual Conference of the International Speech Communication Association, 2012.

[76] Sutton, R.S. and Barto, A.G.: Introduction to Reinforcement Learning. MIT Press, Cambridge, MA, USA, first edition, 1998.

[77] Tan, W.R. *et al.*: Artgan: Artwork synthesis with conditional categorial gans. *Computing Research Repository (CoRR)*, volume abs/1702.03410, 2017.

[78] Urmson, C. *et al.*: Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, volume 25, no. 8, pp. 425–466, 2008.

[79] Versteegh, M. *et al.*: The zero resource speech challenge 2015. In: Interspeech, pp. 3169–3173, 2015.

[80] Villegas, R. *et al.*: Decomposing motion and content for natural video sequence prediction. In: Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[81] Villegas, R. *et al.*: Learning to generate long-term future via hierarchical prediction. *Computing Research Repository (CoRR)*, volume abs/1704.05831, 2017.

[82] Waymo: Technology - waymo, 2017. URL: https://waymo.com/tech/, last viewed on 9 August 2017.

[83] Xie, M. *et al.*: Active and intelligent sensing of road obstacles: Application to the european eureka-prometheus project. In: Computer Vision, 1993. Proceedings., Fourth International Conference on, pp. 616–623. IEEE, 1993.

[84] Yu, F. and Koltun, V.: Multi-scale context aggregation by dilated convolutions. *Computing Research Repository (CoRR)*, volume abs/1511.07122, 2015.

[85] Zhu, J.Y. *et al.*: Generative visual manipulation on the natural image manifold. In: Proceedings of European Conference on Computer Vision (ECCV), 2016.

[86] Ziegler, J. *et al.*: Making bertha drive—an autonomous journey on a historic route. *IEEE Intelligent Transportation Systems Magazine*, volume 6, no. 2, pp. 8–20, 2014.

A. Results of MCNet

This figure shows a high resolution test sample of the trained Motion-Content Network [80] as described in section 4.1.1. The ground truth is placed in the top row while the corresponding prediction can be found beneath. Every time step represents 100ms so that the last frame at $t = 31$ is a prediction of 3.1 seconds in the future.

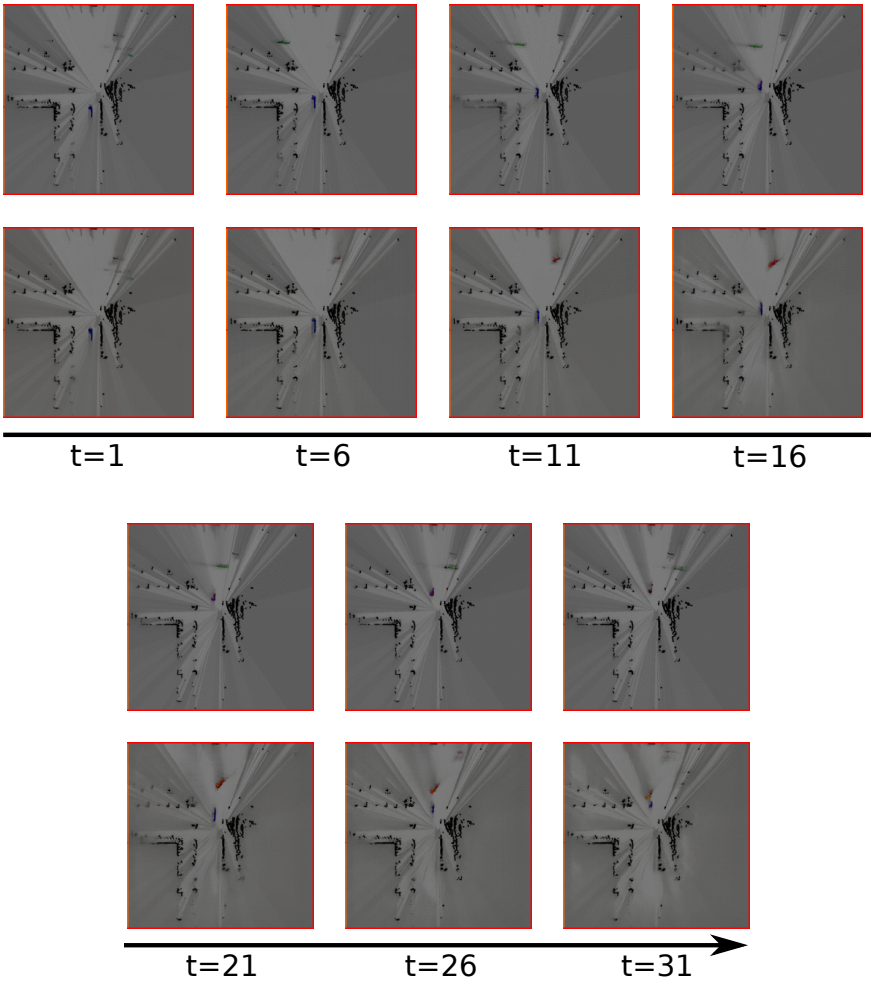


Figure 39: In this intersection scenario a car at the top is waiting for turning left. However another vehicle is crossing its way in the ground truth so that it still has to wait. The network has not seen the crossing car and predicts a turn. This example demonstrates the challenge of predicting unseen objects.

B. Results of prediction network

This section contains further experimental results of section 4.2 including one for map understanding and the other showing the challenge of multimodal situations. The time axis shows the number of predicted time steps where every frame represents 200ms. Ground truth images are placed in the top row while the prediction can be found beneath.

Map understanding and tracking

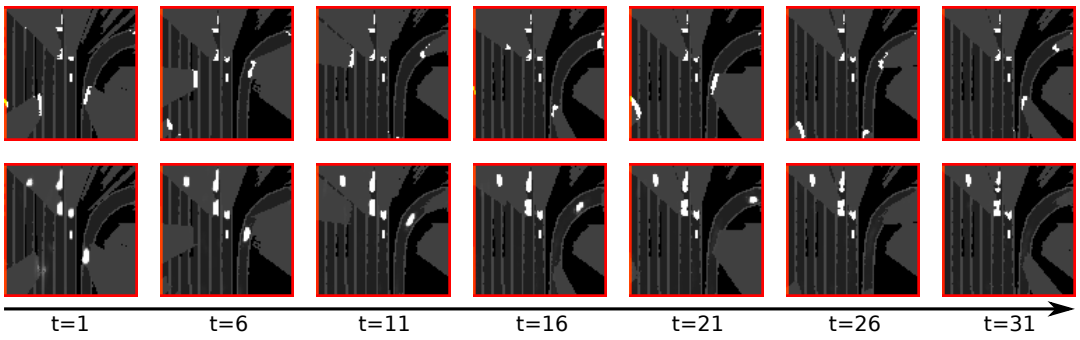


Figure 40: The network can infer trajectories based on the horizon map. Therefore it predicts a certain right turn for the car on the right lane based on the lane structure on which the vehicle drives. In addition the car on the left becomes occluded in the first predicted frames but the network still remembers it at the same position. The ground truth misses this vehicle.

Multimodal scenarios

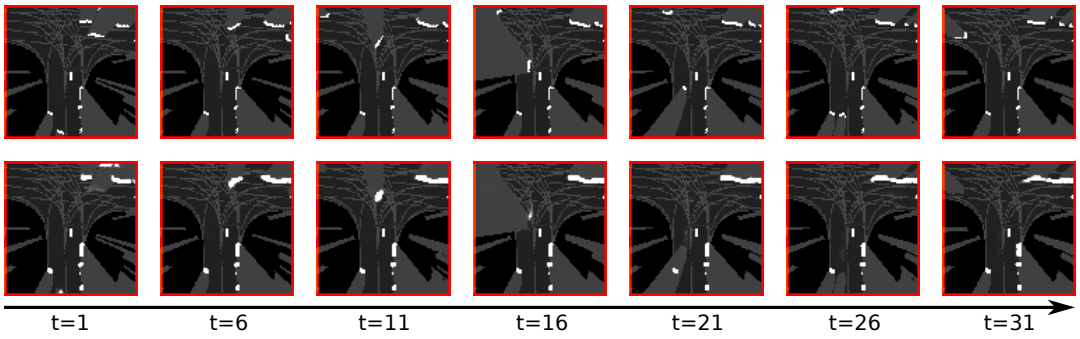


Figure 41: This sequences shows an intersection scenario where the car at the top has multiple possible trajectories. The network predicts a left turn for this vehicle but looses it after about 15 frames because of its uncertainty.Furthermore the other car at the top depends on this trajectory and therefore its position is as well uncertain.