VOLKSWAGEN

AKTIENGESELLSCHAFT

Master Thesis

Maneuver Recognition in Urban Environments with Multi-Stream Networks

Submitted by:
Jana Kirschner

For the Degree of
Master of Science

Dahlem Center for Machine Learning and Robotics
Institut für Informatik, Freie Universität Berlin

First Reviewer:	*Second Reviewer:*	*Volkswagen Contact:*
Prof. Dr. Daniel Göhring	Prof. Dr. Albert Zündorf	Antonia Breuer, M.Sc.
Freie Universität Berlin	Universtität Kassel	Volkswagen AG

19th November 2018

Deklaration

I hereby declare that this master thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institue. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of refereces is given in the bibliography.

Ich versichere hiermit, dass ich die vorliegende Masterarbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe. Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht. Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Berlin, 19th November 2018

Jana Kirschner

FREIE UNIVERSITÄT BERLIN

Abstract

Dahlem Center for Machine Learning and Robotics
Institut für Informatik, Freie Universität Berlin

Master of Science

Maneuver Recognition in Urban Environments with Multi-Stream Networks

by Jana Kirschner

To accomplish the task of highly automated driving it is necessary to get an understanding of the scene surrounding the autonomous vehicle. One step towards the problem of maneuver recognition in the field of intelligent vehicles is closely related to the task of action recognition of human actions. Approaches found in literature solve the problem of action recognition based on RGB and optical flow images for every-day activities. Multi-stream networks with 3D-convolutional layers are frequently used to solve this task. The fusion methods as well as the inputs vary in literature. Besides this, long short-term memory cells are often found in the area of action recognition. Compared to 3d-convolutions they are able to recognize long term motions.
This thesis discusses the question as to whether approaches for action recognition can be used in the field of maneuver recognition. To achieve this, different fusion methods for multi-stream networks as well as different input images were examined. Furthermore, the impact of the addition of a long short-term memory cell was tested and evaluated.

Contents

List of Abbreviations

ADAS advanced driver-assistent system

API application programming interface

BoF bag of features

bbox bounding box

CNN convolutional neural network

CPU central processing unit

DT dense trajectory

FC fully convolutional

FCN fully convolutional network

FV Fisher vector

GPU graphics processing unit

HOF histogram of optical flow

HOG histogram of oriented gradient

iDT improved dense trajectories

LSTM long-short term memory

MBH motion boundary histogram

MLP multi-layer perceptron

NN neural network

NMS non-maximum suppression

ReLu rectified linear unit

RNN recurrent neural network

SIFT scale invariant feature transform

SLP single layer perceptron

SVM support vector machine

w.r.t. with respect to

1 Introduction

This thesis was written in cooperation with the group research of Volkswagen. The aim of this work is to present a possible approach for the task of maneuver recognition in urban environments and to analyze the challenges related to this task. However, this thesis does not provide an universal solution for this problem.

1.1 Motivation

Life is non-negotiable.
Humans are fallible.
Tolerable limits are defined by human physical resistance.
People are entitled to safe transport and safe workplaces. [1, 2]

These four principles describe the fundamental aim of the *Vision Zero* [1, p. 4ff.], [2, p. 2ff.]. Based on the idea that every accident is avoidable it was first formulated in Swedish occupational safety regulation in the year 1811 [1, p. 3]. Currently the Vision Zero is applied to traffic safety. Its ultimate goal is "no fatalities or severe injuries through road traffic crashes" see [3, p. 19].
But we are far from reaching this vision: in the year 2017 approximately 400,000 people got injured and 3,000 died on German roads alone. The reason for 90% of injuries are human errors. Knowing this, it can be concluded that a part of accidents, caused by human errors can be reduced by the help of advanced driver-assistant systems (ADASs) [4].

Besides reducing injuries during individual transportation, another advantage of highly automated driving and driverless cars is, that handicapped and ill people will be able to use individual trasportation. This would improve their mobility and flexibility. Furthermore, people spend on average 1.5 hours per day driving their cars [5]. Through driverless cars it would be possible to use this time e.g., for working or reading. Moreover, it would reduce the overall time spent in vehicles by reducing or even eliminating traffic jams. The authors in [6] determine that equipping 20% of the vehicles on German streets with adaptive cruise control systems would almost eliminate traffic jams. Taking this into account, it is conceivable to reduce traffic jams with driverless cars even more.

1.2 Problem Statement

Maneuver recognition of surrounding objects, such as pedestrians, vehicles and bicycles, plays an important role in the task of highly automated driving. Given a set of inputs, such as sequences of data over time of length T and previously classified objects O, the task of maneuver recognition is to determine the current maneuver $m_o^i(t)$, with i is a unique id, $o \in \{$*passenger car, truck, bus, pedestrian, bicycle*$\}$ and $m \in \{$*crossing, driving, oncoming, parking, preceding, stopping, turn left, turn right, waiting, lane change, parking in, parking out, not on the road*$\}$. The different maneuvers are dependent on the object class. For example a pedestrian either `crosses` the road or is `not on the road` but is not able to do the maneuver class `preceding`.
In the following, we will assume that the inputs consist of sequences of images of length 16, as well as their computed optical flow and their pixel segmented counterpart. The aim of this thesis is to determine whether it is possible to classify maneuvers of surrounding objects under the given assumptions, and which architecture and fusion method is suitable.

1.3 Related Work

Action recognition is a broad research area with a large volume of publications in the field. On a high level of abstraction, action recognition can be categorized as a **classification problem**. In this area many conventional methods are based on the bag of features (BoF) representation, which is a orderless representation of the computed features. Those features are e.g., the result of a scale invariant feature transform (SIFT) [7], which describe scale invariant features. They are extracted from key points in the image generated from Laplacian of Gaussian approximation. After extracting features it assigns an orientation to each of them and only does calculations relative to the key points, which makes this method orientation invariant. Another feature extractor used by Delal et al. [8] is called histogram of oriented gradient (HOG) and extracts features based on the gradient. The features can then be encoded with e.g., the BoF representation, which is the input to an classifier e.g., a support vector machine (SVM), that divides the different dimensions of features into different classes by constructing a set of hyperplanes separating the different classes.
Recently, deep convolutional neural networks (CNNs) achieved state-of-the-art performance on action recognition. Plenty of research groups employ CNNs resembling the following topology. The first convolutional layers can extract features, while the following fully-connected [9] or fully convolutional layers [10] are able to classify the extracted features. The advantage of this approach compared to the ones using a BoF method is

the possibility of a joint optimization of the whole learning architecture. In the last years new approaches came up to solve classification tasks, such as *zero-shot-learning* [11,12] or *adversarial networks* [13], which will only be mentioned briefly but not explored further in this work.

While the majority of approaches do classification based on time series/image sequences, others only use **single images** [14–16]. In general, the **image sequence** approaches can be categorized into how they handle the temporal dimension [17]: *3D-convolutions*, *pre-computed features*, and *temporal dependencies*. A summary of the presented approaches in this thesis, can be seen in Figure 1.1.

The first category uses **3D-convolutions** to extract features along both, spatial and temporal dimensions. Tran et al. [18] introduce an architecture for 3D CNN, which outperforms common 2D CNN architectures and show that small convolutional kernels can achieve better results than large ones. Varol et. al [19] use a 3D CNN to recognize long term temporal motion within 16 to 100 frames. Similar approaches for action recognition with 3D CNN can be found in [20–23]. Next to using convolutional layers in combination with fully-connected layers, it is possible to combine the 3D CNN with long-short term memory (LSTM) and SVM. Lu et al. [24] classify the generated features from 3D CNN with an LSTM.

A different strategy focuses on incorporating **pre-computed features**, such as *optical flow* [25] and *actor's body parts* [26], whereas in many cases the stream's architecture is a 3D-convolutional network. A three-stream architecture introduced by Oliveira et al. [25], combines the position of the actor's body parts, the optical flow and the RGB image. Instead of combining the streams by summing up the scores, which is common for multi-stream networks, the authors combine the different streams by a sequential Markov chain. This approach of integration should enforce the model to learn complementary features in each of the convolutional streams. A different multi-stream architecture combines the RGB image and the optical flow [27, 28]. Khong et al. [27] present an architecture utilizing two streams, that both consist of existing 3D CNN for action recognition [18]. To fuse the different streams, the authors average the two softmax output scores. Another method for combining the streams, mentioned in this paper, is applying a linear SVM after L2 normalization. In comparison to this approach, pre-trained ImageNet models for both streams are used and different fusion methods are compared by Feichtenhofer et al. [28]. Tu et al. [26] introduce a three-stream architecture, which combines three existing two-stream networks for action recognition. As input they use semantic cues, e.g., the detected action region or only parts of it, in combination with optical flow features.

Furthermore, a considerable amount of literature has been published using *temporal dependencies* by combining a **recurrent neural network (RNN)** with a CNN for action

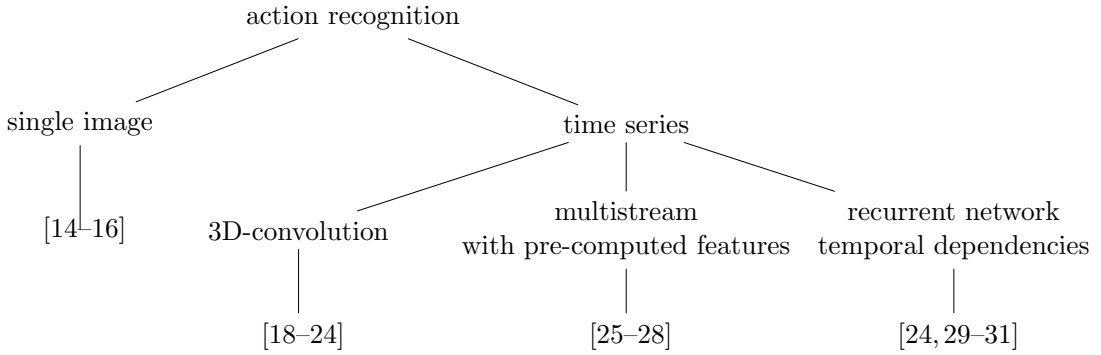


FIGURE 1.1: Taxonomy of different action recognition approaches.

recognition. Common methods apply either action dependent or context dependent features, in which the latter typically correspond to image regions where the action occurs. Sun et al. [29] introduce a two-stream LSTM architecture, referred to as Lattice-LSTM, which learns memory cell transitions independently for individual spatial locations and combines RGB images and optical flow. This architecture can estimate complex motions without significantly increasing the model complexity. Another two-stream architecture can be found in [30]. Approaches based on a 3D-convolutional networks are proposed in [24, 31], here the approach of Lu et al- [24] is used to *detect* a specific action of `falling`.

In the area of **maneuver recognition/prediction** common approaches often use methods like SVMs [32], Bayesian Networks [33], and Hidden Markov Models [34] for classification based on motion features such as speed and yaw rate. A different approach [35,36] classifies driving maneuvers based on prototype trajectories which are computed in advance. Dinstiguished from these classic methods Lenz et al. [37] introduce a feature-based deep neural network which is used to parameterize a Gaussian Mixture Model. This architecture can predict the probability distribution over all possible actions for a vehicle. Compared with action recognition approaches, which are mainly based on image inputs, the maneuver recognition methods mentioned above are in general based on pre-computed input features.

This thesis focuses on maneuver recognition by using a multi-stream convolutional network with 3D convolutions. In the following we will use the term `action` in the UCF dataset, while we will use the term `maneuver` in the context of automated driving for both human motion an vehicle motion. Possible maneuvers are for example `preceding` and `lane change`.

To fulfill the task of maneuver recognition based on convolutional networks, this thesis is based on the approach described in [19]. It was chosen because of its good results on the UCF101 dataset. On top of that it proposes different methods to improve results, which is often neglected in other approaches. The aim of this thesis is to validate our

proposed algorithm on the introduced urban scenario data set, rather than improve the human action recognition data set UCF101.

1.4 Description of Contents

The thesis is organized as follows: First an introduction of the fundamentals is given in Chapter 2. Subsequently, the paper on which this work is based on, and the methodology is explained in Chapters 3 and 4. The dataset preprocessing is outlined in Chapter 5. Subsequently, in Chapter 6 the task of maneuver recognition on a dataset of tactical driving maneuvers is evaluated. To fulfill this task, different architectures and parameters are tested. The thesis concludes with a critical reflection of this work and an outlook into future research in Chapter 7.

2 Fundamentals

This Chapter gives an overview of the fundamental concepts used for this thesis. Detailed information can be found in literature. In the following, artificial networks theory will be outlined in Chapter 2.1, then Chapter 2.2 describes the functionality of convolutional neural networks (CNNs). Since this thesis deals with multi-stream networks, Chapter 2.3 will explain the fundamentals of the fusion of different neural network streams. Chapter 2.4 describes long-short term memory cells and Chapter 2.5 introduces the optical flow, which can be used to extract motion in image sequences.

2.1 Artificial Neural Networks

Artificial neural networks were first mentioned in [38]. The very simple model of neurons proposed by McCulloch in [38] has only binary inputs and binary outputs and can compute logical computations such as ADD and OR. In 1958 Rosenblatt [39] invented a new model of neural network (NN), which is called *perceptron*. This perceptron is the basis of many NN and deep learning models. Since then, many NN architectures have been used for different practical applications, e.g., classification and forecasting [40].
The general idea of NNs is based on the human brain. "With respect to processing information the neurons are the most important components of the nervous system." [41]. There are about 100 billion neurons in the human brain, of which a large part can act in parallel. The information processing is done by interaction of different neurons through electrical activity. Each neuron is surrounded by a cell body, that has long extensions, called axons as well as short extensions, called dendrites. A place where an axon and a dendrite almost touch, is called synapse. This place is where the activation potential is given to the next neuron. An information is encoded by the electrical potential, as well as the impulse that is transmitted per second [41].

The simplest model of a NN is called an single layer perceptron (SLP), which consists only of input and output neurons as depicted in Figure 2.1. The neuron is connected to n inputs by its incoming edges x_i, that are weighted with a factor w_i and summed up with the input function. The addition of the bias to this sum is similar to the threshold in a

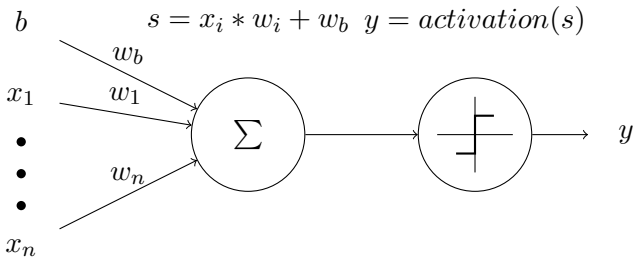


FIGURE 2.1: Network architecture of a simple single layer perceptron.

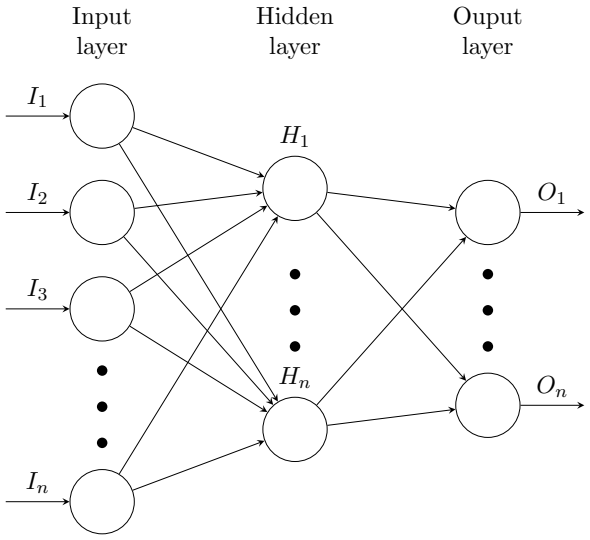


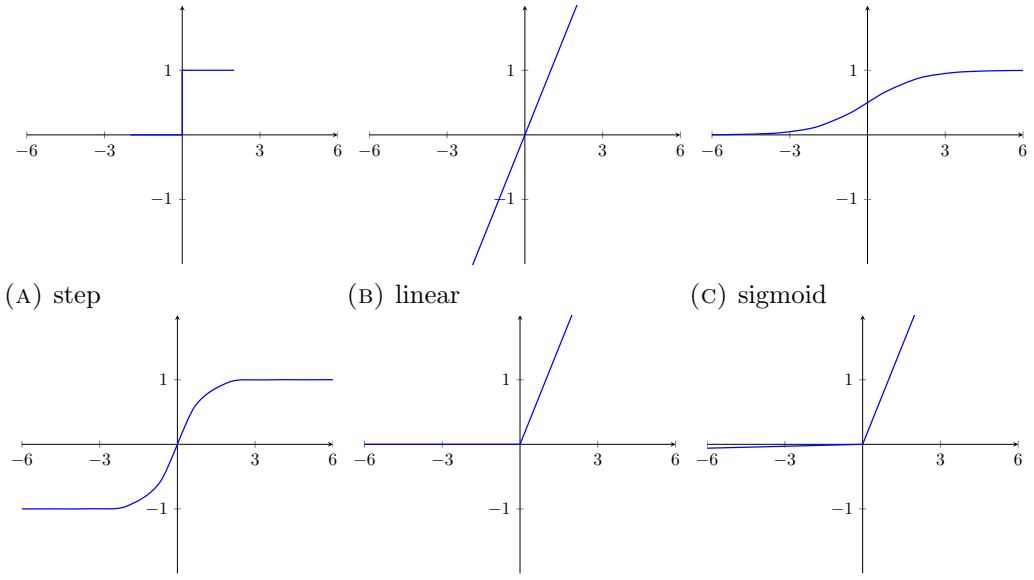
FIGURE 2.2: Network architecture of a simple multi layer perceptron.

human neuron, in which a neuron only fires if passing a certain threshold. Considering the approximation of a linear function, the bias is represented by b in $y = ax + b$ [41]. After summing up the inputs, the activation function determines the overall output of the neuron, called activation. In the case of an SLP e.g., for a binary classification, the activation function determines whether the output is true or false. In the context of a network with one or more hidden layers (a multi-layer perceptron (MLP)), this could also be one of the input edges of another neuron, see Figure 2.2 .

Considering an MLP, there are three basic types of *layers*, see Figure 2.2: *input layer*, *hidden layer*, and *output layer*. The neurons of each layer k are connected to all neurons in the layer $k + 1$. This configuration is called *fully-connected*.

Activation Functions

Basically each neuron can output values in the range from negative infinity to positive infinity. The decision of whether a neuron is activated or not is made by activation functions.
The simplest form of an activation function is the **step function** which outputs 1 if the neuron's output value is greater than a specific threshold, else 0, see Figure 2.3(A). In case of this activation function, one neuron can only be completely activated or not at all, which is in general not useful for e.g., multi class classification. There multiple neurons should be activated, for outputting a certain score for each class.
A **linear** activation function, which passes the input without any changes is still a very simple model, see Figure 2.3(B). If more than one neuron is activated the output class will be based on a defined rule e.g., the maximum activation. But there is still a drawback. Since the derivative is constant, multiple layers are equivalent to a single layer neural network [42]. Another problem with linear functions is, that they do not limit the neuron's output, which means that the values can get very high [42].
With the use of a **sigmoid** activation function, it is possible to limit the neuron's output to the range from 0 to 1. Another advantage is its non linearity. If multiple layers with linear activation functions are put together, the result of the last layer will always be a linear function of the input of the first layer. Which means that these networks can be reduced to a single layer network. This is no longer the case by using a sigmoid function or any other non linear function. But since the function is flat near 0 and 1 the gradients can become very small. This prevents the network from training [42], see Figure 2.3(C) for a plot of the function.
In Figure 2.3(D) a **scaled sigmoid** function can be seen. This means the gradients are stronger than in Figure 2.3(C), but still on either end of the function, the *vanishing gradient* problem is present.
The **ReLu** function is nonlinear and not continuously differentiable, its outputs are between 0 and positive infinity. Since this activation function can simply be implemented by thresholding it is very efficient. As the output in its positive region is equal to the input, it solves the vanishing gradient problem in this region [42], see Figure 2.3(E).
By changing the negative part of the activation function it is possible to mitigate the vanishing gradient problem. The **leaky ReLU** for example multiplies the input with a constant small term e.g., 0.01 in the negative region of the function [42], see Figure 2.3(F). Finding a suitable activation function is still an open topic in research.



(A) step (B) linear (C) sigmoid

(D) tanh (E) RELU (F) leaky RELU

FIGURE 2.3: Visualization of different activation functions.

2.1.1 Training

The objective of training a network is to minimize the loss on the training data set, as well as generalization. The loss can be described as $L(a, (\mathbf{x}, \mathbf{t})) = ||a(\mathbf{x}) - \mathbf{t}||^2$, where a represents an activation function, x an input vector and t the target vector. A common method for training is the backpropagation algorithm. The main idea of backpropagation is to determine the errors in the hidden units (which cannot be computed directly) in an MLP by backpropagating the errors through the network [43, p. 27 ff.]. The training procedure is divided in the *forward pass* and *backward pass*.

Backpropagation

The basic algorithm loop can be structured as follows [40, p. 102 ff.]:

1. Present the pattern at the input layer.
2. Let the hidden units evaluate their output using the pattern.
3. Let the output units evaluate their output using the result in step 2 from the hidden units.
4. Apply the target pattern to the output layer.
5. Calculate the δs on the output nodes. Which is defined by $\delta_k = \sigma'(a_k) \sum_{j \in I_k} \delta^j w_{jk}$, with σ' describing the gradient of the activation function a of node k, I_k the k^{th}

input node and w_{jk} the trainable weight between node j and k. $\delta^j w_{jk}$ can be written as $\eta(t_j - y_j)g'(h_j)x_i$, with t_j describing the target value and y_j the actual output. g' is the derivative of the activation function σ, $x_i = \frac{\delta h_j}{\delta w_{jk}}$, h_j the neuron's input and η the learning rate.

6. Train each output node using gradient decent.
7. For each hidden node, calculate its δ.
8. For each hidden node use the δ found in step 7 to train according to gradient descent.

The steps 1-3 are known as forward pass and steps 4-8 as backward pass. During the forward pass the output of each neuron is evaluated with respect to its activation function as well as the output of the neurons in the previous layers [43, p. 27 ff.]. In the following, the backpropagation algorithm will be explained based on [43, p. 27 ff.] and [40, 102 ff.]. The output of a neuron is determined as:

$$y_k^p = F(s_k^p), \tag{2.1}$$

in which

$$s_k^p = \sum_j w_{jk} y_j^p + b_k, \tag{2.2}$$

where F is the chosen activation function, y_k^p is the output with respect to an input pattern p of a neuron k. The summation is done over nodes j connected to node k.
To measure the error at the output layer, the chosen error function E^p is applied e.g., the total quadratic error is defined by:

$$E^p = \frac{1}{2} \sum_{o \in N_o} (t_o^p - y_o^p)^2, \tag{2.3}$$

where t_o^p is the desired output for the layer and y_o^p is the calculated output. The delta of each weight in the network is calculated by

$$\Delta_p w_{jk} = -\eta \frac{\delta E^p}{\delta w_{jk}}. \tag{2.4}$$

By using the delta of each weight in the network, the backpropgation algorithm is minimizing the error function E^p

$$\frac{\delta E^p}{\delta w_{jk}} = \frac{\delta E^p}{\delta s_k^p} \frac{\delta s_k^p}{\delta w_{jk}}. \tag{2.5}$$

The second factor is written from Equation 2.2 as

$$\frac{\delta s_k^p}{\delta w_{jk}} = y_j^p. \tag{2.6}$$

We define

$$\delta_k^p = \frac{\delta E^p}{\delta s_k^p}. \tag{2.7}$$

The update rule for each weight can be described as

$$\Delta_p w_{jk} = -\eta \delta_k^p y_j^p, \tag{2.8}$$

with η describing the learning rate which defines the step size. To compute δ_k^p we apply the chain rule and now have a product of two factors,

$$\delta_k^p = -\frac{\delta E^p}{\delta s_k^p} = -\frac{\delta E^p}{\delta y_k^p}\frac{\delta y_k^p}{\delta s_k^p}, \tag{2.9}$$

by computing the second factor of Equation 2.1 it yields

$$\frac{\delta y_k^p}{\delta s_k^p} = F'(s_k^p), \tag{2.10}$$

which simply is the first derivative of the activation function. Starting from the output layer we get

$$\frac{\delta E^p}{\delta y_k^p} = -(t_o^p - y_o^p), \tag{2.11}$$

after substitution of Equation 2.10 and 2.11 in Equation 2.9:

$$\delta_o^p = (t_o^p - y_o^p)F_o'(s_o^p) \tag{2.12}$$

If k is not an output layer we have no information about the contribution of one hidden layer to the output error. To solve this problem we need a function that computes the error for the hidden layer up to the output layer. To compute this we can write:

$$\frac{\delta E^p}{\delta s_k^p} = \sum_{k=1}^{N} -\frac{\delta E^p}{\delta s_o^p}\frac{\delta s_o^p}{\delta y_k^p}, \tag{2.13}$$

after substitution of $\delta_o^p = -\frac{\delta E^p}{\delta s_o^p}$

$$\frac{\delta E^p}{\delta s_k^p} = \sum_{k=1}^{N} \delta_o^p \frac{\delta s_o^p}{\delta y_k^p}, \tag{2.14}$$

by substituting s with Equation 2.2 $s_k^p = \sum_j w_{jk} y_j^p + b_k$:

$$\frac{\delta E^p}{\delta s_k^p} = \sum_{k=1}^{N} \delta_o^p \frac{\delta}{\delta y_k^p} w_{k,o} y_k^p = -\sum_{k=1}^{N} \delta_k^p w_{k,o} \tag{2.15}$$

after substituting this in Equation 2.9 and 2.10 it yields:

$$\delta_k^p = F'(s_k^p) \sum_{k=1}^{N} \delta_o^p w_{k,o}. \tag{2.16}$$

Equations 2.13 and 2.15 are computed recursively for each layer in the network, the results are used to compute the weight changes according to Equation 2.9.
According to step 8, the gradient descent algorithm adds the calculated weight changes to the previous weights in the network and thus improves to network's classification quality.

2.1.2 Gradient Descent Optimizer

As can be seen in the previous section, the updating of each hidden node is achieved by the gradient descent method. Apart from this basic approach of gradient descent there are different variants of this algorithm.
For simplification the gradient descent algorithm $\Delta w_{ij}(t+1) = -\eta \delta_k^p y_j^p + \Delta w_{jk}(t)$ is substituted by $\theta = \theta - \eta \delta_\theta y(\theta)$, where θ is a set of parameters describing the network e.g., the weights of the layers.

Batch Gradient Descent

The batch gradient descent computes the gradient with respect to (w.r.t.) the parameters θ of the entire training data set:

$$\theta = \theta - \eta \delta_\theta y(\theta). \tag{2.17}$$

This procedure needs just one update and can only be calculated for data sets, which fit in the memory, but it converges to a local minimum [44].

Stochastic Gradient Descent

Stochastic gradient descent performs parameter updates for each sample in the training data individually; that is a forward pass as well as a backward pass with a parameter

update performed for a single sample:

$$\theta = \theta - \eta\delta_\theta y(\theta; x^{(i)}; y^{(i)}). \tag{2.18}$$

This algorithm can be used in online training and is much faster than batch gradient descent. Batch gradient descent converges to the local minimum if there are enough different samples in one batch but stochastic gradient descent often jumps around different local minima [44].

Mini-batch Gradient Descent

The third variation of gradient descent is called mini-batch gradient descent and performs one backward step for each mini-batch. One mini-batch includes multiple samples from the training data set:

$$\theta = \theta - \eta\delta_\theta y(\theta; x^{(i:i+n)}; y^{(i:i+n)}). \tag{2.19}$$

This variation reduces the amount of parameter updates, which can prevent jumping between small local minima and leads to a more stable convergence [44].

The three algorithms of gradient decent do not deal with finding an appropriate learning rate and are not suitable for highly non-convex error functions [44]. In the following gradient descent techniques that are widely used for deep learning, will be outlined.

Momentum

While Gradient Descent is likely to oscillate across the slopes of a local minimum, momentum helps to accelerate in the relevant direction and dampens oscillations. This is done by adding a fraction from the update vector of the last time step [44].

$$v_t = \gamma v_{t-1} + \eta\delta_\theta y(\theta) \tag{2.20}$$

$$\theta = \theta - v_t \tag{2.21}$$

Nesterov accelerated gradient

By investigating the error function at the next step's value it is possible to get an idea what the error function will look like in the next steps. While the Momentum algorithm jumps in the direction of the new gradient, Nesterov accelerated gradient first jumps to the direction of the previous gradient, then calculates the gradient and does a correction

of its step. With this technique, it is possible to locate a change of direction in the error function [44].

$$v_t = \gamma v_{t-1} + \eta \delta_\theta y(\theta - \gamma v_{t-1}) \tag{2.22}$$

$$\theta = \theta - v_t \tag{2.23}$$

Adagrad

The previous optimizers only adapt the updates to the slope of the error function. Adagrad adapts the learning rate to the parameters to perform smaller or larger updates. The updates depend on the occurrence of associated features.
Adagrad computes the gradient at time step t w.r.t. parameter θ_i as follows:

$$g_{t,i} = \gamma_\eta y(\theta_{t,i}). \tag{2.24}$$

Consequently the parameter update can be computed by:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} * g_{t,i}, \tag{2.25}$$

$G_t \in \mathbb{R}^{d \times d}$ is a diagonal matrix, where the i-th element on the diagonal is the sum of squares of the gradients w.r.t. θ_i up to time instance t. To avoid division by zero $\epsilon \in \mathbb{R}$ is added, which is close to zero but larger than zero.
Adagrad's weakness is that the learning rate can get very small, because of the addition of squared gradients in the denominator, which are always positive [44].

Adadelta

Adadelta tries to overcome the monotonically decreasing learning rate with a sliding window for adding the last w gradients, instead of adding all last gradients. This is done by a running average approach $E[g^2]_t$ for each time step [44].

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{E[\sqrt{g^2}] + \epsilon} * g_{t,i}. \tag{2.26}$$

Adam

Adaptive Moment Estimation (Adam) is another approach to compute adaptive learning rates for each parameter. Additionally to storing the average of past squared gradients

v_t, it stores an average of the past gradients m_t [44].

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \tag{2.27}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2. \tag{2.28}$$

Since the authors of Adam observed that m_t and v_t are biased towards zero, they used a new bias-corrected estimation:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \tag{2.29}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}. \tag{2.30}$$

The Adam update rule is computed as follows:

$$\theta_{t+1} = \theta - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t. \tag{2.31}$$

2.1.3 Regularization

A central challenge in deep learning is to make the network perform well on both the training data as well as unknown inputs. Strategies to reduce the test error are known as regularization. By observing the training and validation error it is possible to estimate whether the network is overfitting or underfitting. By achieving good generalization on the training data set the model will neither overfit nor underfit. A visualization of this effect is provided in Figure 2.4. In the following some regularization techniques are outlined.

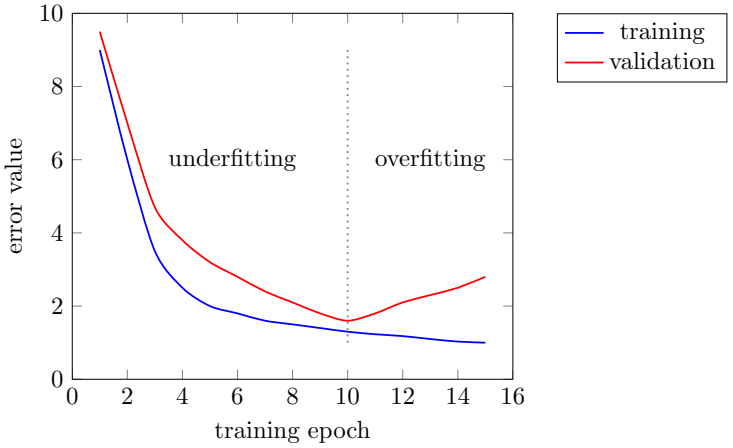


FIGURE 2.4: Underfitting and overfitting during training.

Parameter Regularization

Weight decay is one of the most common regularization techniques. By adding a regularization term, it drives the weights closer to the origin. But regularizing the parameters to be near any point would still yield a positive effect. [45, p. 116 ff., p. 126 ff.] For L2 regularization this is done by adding the term $\Omega(\theta) = \frac{1}{2}||w||_2^2 = \sum_i |w|_i^2$, whereby w denotes all weights that should be affected by regularization and θ indicates all parameters. While L2 regularization is the most common form, there are other ways, like L1 weight decay, computed as $\Omega(\theta) = ||w||_1 = \sum_i |w|_i$ [45, p. 116 ff., p. 126 ff.].

Data Set Augmentation

Another way to make a model generalize better is training it on more data. Since in most cases, the amount of data is limited, one way to get around this problem is data set augmentation. Creating new fake data can be done by transforming the training data. In case of images as input data operations like moving images by some pixels in each direction often improves generalization greatly [45, p. 136 ff.]. But operations like rotation, scaling, mirroring and injecting noise are also common.
However, transformations should be applied carefully, since the class must stay the same. The class changes by accidentally transforming the maneuver `left-turn` to the maneuver `right-turn` by flipping the image.

Early stopping

By training large models which are likely to overfit, it could be helpful to stop training before the error has decreased too much. A small error value describe a good adaption of the parameters for the training dataset. If the parameters fit perfectly to the training dataset, the performance for the testing dataset suffers. By observing the training and validation error, the model will be saved each time the validation error starts increasing again. After the training has stopped, the saved parameters will be used rather than the latest parameters. If none of these saved parameters improve, the training process terminates. [45, p. 241 ff.]

Dropout

The last technique presented here is applying dropout to specific layers to prevent overfitting. The idea of dropout is to randomly drop neurons in specific layers. This creates smaller subnets which are more likely to generalize than big ones [46].

2.2 Convolutional Neural Networks

For image processing a special kind of network called convolutional neural network (CNN) is used. The approach is inspired from the brain's visual cortex, in which multiple visual neurons have a *local receptive field*, which means that they react to stimuli in this field. With the use of receptive fields in neural networks, there are considerably less parameters needed for a CNN, than for a fully-connected network [47]. In fully connected networks, weights connect each input value to each neuron in the first layer. CNNs only use one weight for each receptive field. Using a fully-connected network for image inputs, one weight for each pixel value would be needed.

Convolutional Layer

The convolutional layer is based on the convolutional operation from mathematics. It slides a function (kernel) over another and calculates the integral of a pointwise multiplication [45, p. 327 ff.]. When applying this on images, the output is a picture with specific features from the original picture. For a two-dimensional image I and a two-dimensional kernel K, the convolutional operation can be shown as:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n). \tag{2.32}$$

The kernel k denotes a multidimensional array with parameters that are changed during the learning process [45]. An example of the convolutional process can be seen in Figure 2.5.

Each parameter in the kernel is a weight in the neural network. Comparing the different kernels that are used in the training process to the input size, which is the image size, the CNN has much less parameters than a fully-connected network. Since the input is in general an RGB-image, the kernel consists of three kernel levels, one for the red image, the green image and the blue image.
The sliding procedure is specified by *padding* and *striding* parameters. Stride controls the step size of the convolving kernel. In some cases there are multiple rows and columns left, because the kernel does not exactly fit multiple times into the input volume. In this case, it is possible to pad zeros around the input, so that the kernel fits exactly.
In the common architecture of a convolutional layer, an activation function and a *pooling* layer follows the convolutional operation, which will be introduced subsequently.

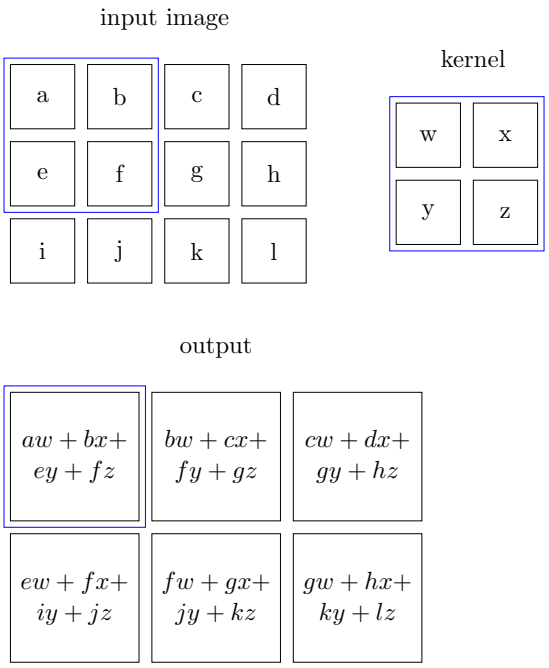


FIGURE 2.5: Convolutional operation with stride one, see [45, p. 330]

.

Pooling

Generally, to subsample the output of a convolutional layer, a pooling layer follows. This further reduces the number of parameters compared to a fully-connected neural network (NN). Similar to convolutions there is a kernel, which slides over the image. The sliding procedure is specified by *padding* and *striding* parameters, similar to the convolutional layer.

There are different pooling approaches, such as *max pooling* and *average pooling*. While, max pooling takes the maximum value, which is covered by the filter, average pooling calculates the average value. An example of max pooling can be seen in Figure 2.6 [45, p. 335 ff.].

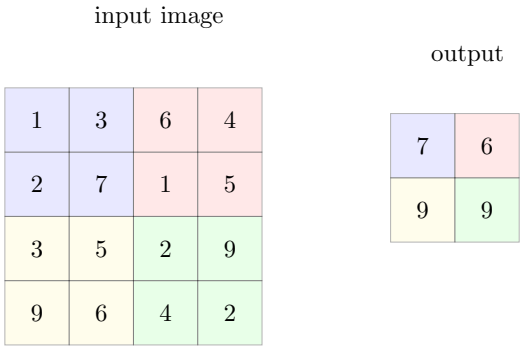


FIGURE 2.6: Max pooling operation with 2×2 filter and stride two.

3D-Convolution

Additional to convolution and pooling in the spatial axis, it is possible to do so in the temporal axis on image sequences. In this case the kernels do not only slide over one image, but slide over one image and over the temporal axis. With this approach it is possible to recognize spatial, as well as temporal features.

2.3 Multi-Stream Neural Networks

Convolutional networks in general handle only one data stream, which is commonly an RGB image. Multi-stream networks can process multiple input streams. This has the advantage that for e.g., action recognition it is possible to feed movement related data, such as optical flow.
To fuse the streams, there are different possibilities. Karpathy et al. [48] show different variants of fusion methods for temporal information. While *early fusion* combines the information on the pixel level, at the first convolutional layer, *late fusion* processes the information through two different neural network streams and fuses it in the fully connected layer. The combination of both approaches is called *slow fusion*, which fuses the information piecewise throughout the network, such that higher layers have more information than lower layers.
Karpathy et al. [48] did not investigate different input types like optical flow and RGB. Simonyan et al. [23] for example use these fusion methods on two-stream networks with RGB and optical flow inputs. They try two different late fusion approaches, which are averaging and classification via support vector machine (SVM). In the former approach the authors average the output of each network and take the softmax. In the latter approach an SVM takes the softmax scores of the different streams as feature inputs.
On top of to finding the best appropriate layer in the network to fuse, which was discussed above, Feichtenhofer et al. [22] analyzed the fusion method itself. They investigated five different approaches, the first being the sum of the two feature maps, this method does not define a correspondence between the pixels of each layer. Similarly to this, taking the maximum of each feature map has the same drawback. By concatenating the maps there is still no correspondence between pixels. However, this can be achieved through an extension called convolution fusion. In this approach the maps are first concatenated and then convolved. This reduces the dimensionality by the factor of two and enables learning the correspondence of the pixels in each stream. The last presented approach is called bilinear fusion and first computes the outer product of the two feature maps and then sums up the features of each location.

2.4 Long-Short Term Memory

As 3D-convolutions only capture motion patterns in the time frame of the input stream, long-term temporal motions which are larger than the input stream, can be recognized by long-short term memory (LSTM) cells [23]. While recurrent neural networks (RNNs) suffers from vanishing and exploding gradients, LSTMs solves this problem by using several learning gates. In the following an exemplary walk through a LSTM cell is shown, based on [49], [50] and [51].

While traditional neural networks cannot remember past information, RNN have loops, allowing information to persist. Each LSTM unit contains one or more memory cells and three gating units, which are called *input*, *output* and *forget*. The cell's recurrent unit is called *Constant Error Carousel*, which provides short-term memory storage by recirculating the activation and error signals. By training the gates, it can be learned which information to store, how long to store it and when to read new information, see Figure 2.7.

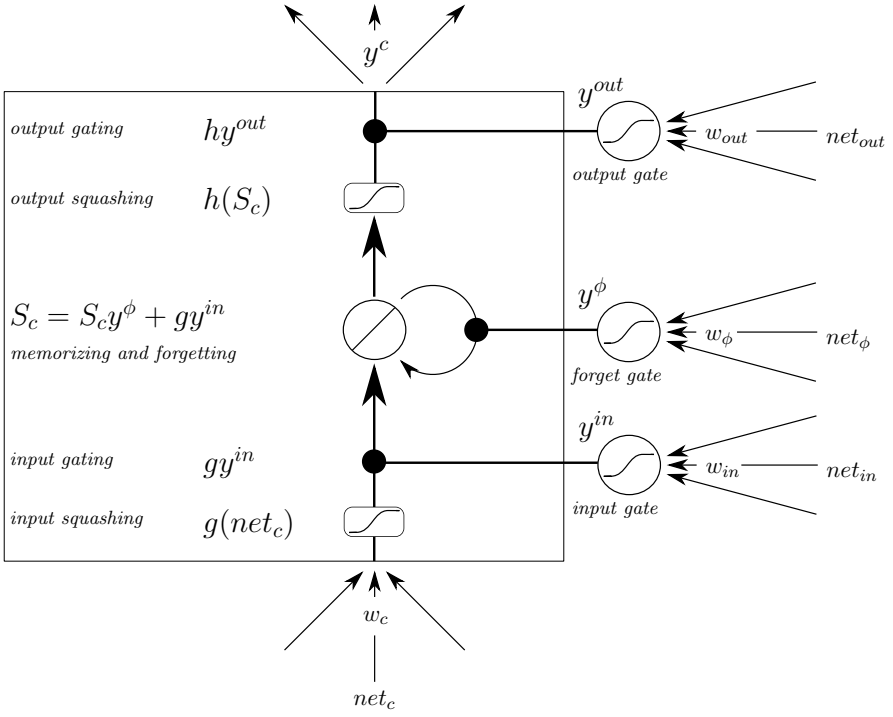


FIGURE 2.7: LSTM cell, with three different gates, see [50].

If a gate is closed, information and noise cannot enter the cell. The update of the cell state is based on three different inputs, the input to the cell itself net_c and the inputs to output gate net_{out}, forget gate net_{in}, and input gate net_{ϕ}.

Information flowing through the cell first gets squashed by an input activation function. Then the input gate controls, whether information can pass or not. If no information

passes the cell the cell state is not updated. Similar to this the output gate controls the information flowing out of the cell.

The role of each gate is defined in Equation 2.33 - 2.37. If i_t is activated the the oncoming information will be processed. Respectively, if f_t is activated the last cell status c_t will be forgotten. The output gate o controls whether the current cell output will be propagated to the hidden state h_t.

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \tag{2.33}$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \tag{2.34}$$

$$c_t = f_t \circ c_{t-1} + i_t \circ tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{2.35}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co} \circ c_t + b_o) \tag{2.36}$$

$$h_t = o_t \circ tanh(c_t) \tag{2.37}$$

One major drawback of the classic LSTM presented above is the handling of spatiotemporal information. Before putting data into the LSTM it has to be flattened, which looses all spatiotemporal information. For this reason Shi et al. [52] introduce a convolutional LSTM. In this layer, all the inputs, outputs, states and gates are $3D$ tensors with spatial dimensions in the last two dimensions. For a better understanding Shi et al. explain these dimensions as feature vectors standing on a spatial grid. The convolutional operations enable the possibility of determining the future state not only on the past state but on the local neighbors. The authors use the convolutional operations in the *state-to-state* and *input-to-state* transitions, see Equations 2.38 - 2.42.

$$i_t = \sigma(W_{xi} * x_t + W_{hi} * h_{t-1} + W_{ci} \circ c_{t-1} + b_i) \tag{2.38}$$

$$f_t = \sigma(W_{xf} * x_t + W_{hf} * h_{t-1} + W_{cf} \circ c_{t-1} + b_f) \tag{2.39}$$

$$c_t = f_t \circ c_{t-1} + i_t \circ tanh(W_{xc} * x_t + W_{hc} * h_{t-1} + b_c) \tag{2.40}$$

$$o_t = \sigma(W_{xo} * x_t + W_{ho} * h_{t-1} + W_{co} \circ c_t + b_o) \tag{2.41}$$

$$h_t = o_t \circ tanh(c_t) \tag{2.42}$$

2.5 Optical Flow

As stated by Baker et al. [53] the "optical flow is the apparent motion of the brightness patterns in the image". In other words, the goal of optical flow is to calculate the

trajectory of each pixel over time. This task poses difficulties in realistic videos e.g., illumination changes and large displacements.

There are two popular approaches by Lukas/Kanade [54] and Horn/Schunck [55]. In the following, both approaches will be explained briefly. In general there are several assumptions used to simplify the optical flow calculation, such as brightness constancy and temporal consistence [56]. From those, it follows that the motion constraint can be written as $\nabla I^T \mathbf{v} + I_t = 0$, where $\mathbf{v} = (u, v)^T$ represents the optical flow with its horizontal and vertical velocities, $\nabla I = (I_y, I_y)^T$ its intensity gradient and I_t is the temporal derivative at inference time t [56]. As can be seen, this equation is an ill-state problem, because it contains two unknowns. There are different approaches to overcome this problem, which are mainly known as *local* and *global* approaches.

Lukas Kanade et al. [54] use the spatial coherence assumption, which means that neighboring pixels move similarly and because of that share the same flow. This follows the assumption that pixels on the same surface are likely to move together [56]. This technique can be classified as a *local* method and is calculated by

$$min_{u,v} E_{LK} = \sum_{i=1}^{3} G_\sigma * [I_{ix}(\mathbf{x}).u + I_{iy}(\mathbf{x}).v I_{it}(\mathbf{x})]^2, \tag{2.43}$$

where G_σ is the Gaussian convolution with derivative σ. I is the spatial and temporal derivative for each of the three channels in the RGB image.

Horn and Schunck use a *global* technique to overcome the ill-state problem of the optical flow equation, by applying a smoothness term. It can be calculated by

$$min_v E_{HS} = \int_I [\sum_{i=1}^{3} \nabla I_i(x).\mathbf{v} + \lambda |\nabla \mathbf{v}|^2] dx dy, \tag{2.44}$$

where λ is the regularization term and $|\nabla \mathbf{v}| = |\nabla u|^2 + |\nabla v|^2$ [56]. The advantage of Horn and Schunck is the smoothness term. Due to $\lambda |\nabla \mathbf{v}|^2$ it is possible blurring the flow at the motion boundaries [56].

Alongside these geometrical approaches there are neural network based approaches as well. For example Ilg et al. [57] do an end-to-end optical flow estimation with a convolutional neural network (CNN). They combine different FlowNets to accomplish the task of large and small displacement optical flow estimations. While large displacements in general belong to fast movements, small displacements correspond to slow actions.

3 Methodology

This chapter focuses on the used toolboxes for machine learning and the workflow of the implementation. The toolboxes were chosen to ensure a fast and simple integration into the autonomous vehicles. Further the basic tool setup will be introduced, which includes the used graphic cards and workstation setup.

3.1 TensorFlow

"TensorFlow is an interface for expressing machine learning algorithms, and an implementation for executing such algorithms." [58]. It was developed by Google Brain and is their second generation of a machine learning interface after DistBelief.
TensorFlow provides a Python and a C++ interface. This thesis is based on the Python frontend. All computations in TensorFlow are described by a directed graph, the values that flow along the edges are *tensors*, which are multidimensional arrays. The element type is specified at graph construction time.
To run computations it is necessary to create a *session*, which makes the client program interact with the TensorFlow system. A primary operation of a session is *Run*. It computes outputs of given variables with a given set of tensors.
It is possible to put a graph on different devices. Each node has to be placed on a device, such as a graphics processing unit (GPU) or a central processing unit (CPU). After the manual construction of the graph, TensorFlow replaces each edge between two devices with special *Send*, and *Receive* node, which coordinate the data transfer between the different devices [58].
Next to the built-in device communication, TensorFlow has several useful functions for training and testing a neural network. A very useful functionality is *gradient computation*. If a tensor C depends on a set of other tensors X_k, then dC/dX_k can be computed. For doing this, TensorFlow first finds a path from X_k to C and then does backtracking to X_k. By doing this, for every node in the graph, it adds a new one, containing the partial gradients. The newly added nodes can now be used as a *gradient function* by any other operation [58].
Another useful feature is that TensorFlow provides partial execution of the graph [58].

This allows the user to only run subgraphs of the built graph by calling the node's name. Besides these basic TensorFlow functions, different application programming interface (API)s are provided e.g., for different layer computations, such as *convolutional layer* and *pooling layer*, for a multi-threaded input pipeline, as well as *tensor board* for supervising the training and testing procedure.

3.2 Tool Setup

This thesis' practical part was mainly developed in Python. As mentioned in the section before, TensorFlow was used for training and testing the neural networks. The models were trained by utilizing a graphics processing unit (GPU). While the single-stream networks fit on one GPU, multi-stream networks only fit on multiple GPUs, one for each stream.
The models were trained on a Nvidia GeForce GTX 1080 with 11GB memory. The code is tested on an Ubuntu 18.04 LTS system, with Python 3.6.5 in combination with TensorFlow 1.7, cuda 9.0 and libcudnn 7.0.

TensorFlow provides different possibilities to design the network as well as the data pipeline. By using these extensions dataset handling can become much easier. In this thesis the choice of the input pipeline was in consideration of using TensorFlow-records, using dataset API and writing an own data pipeline.
The decision for using the dataset API was based on the fact that it can shuffle the dataset as well as the capability of parallelized fetching of input data. In this thesis the data is not read from TensorFlow-records, but from disk. This is advantageous, because the datasets change quickly and there is no need to write the whole TensorFlow record file again.

3.3 Implementation Workflow

The implementation can be split into two different parts. The dataset generation and the TensorFlow computations, as can be seen in Figure 3.1. Images of recorded vehicles drives are stored in a folder structure of *sampeled dataset - class name - id*. With this structure it is possible to review the dataset easily. Moreover, the multiple folder structure prevents the explorer from crashing through too many files within a single folder. For the multi-stream network it is necessary to generate optical flow images as well as pixel segmented images for each used dataset. To generate a new dataset, this procedure has to be repeated for the RGB images as well as for the optical flow and pixel wise segmented images.

After generating the dataset, the model has to be built. This is done with the use of TensorFlow. For this thesis five different models are built. To feed the data into the models the TensorFlow API was used. With the use of this API it is possible to shuffle and batch the data set easily. Depending on the used model the training procedure was implemented. For models including long-short term memory (LSTM)s the training method is different to sequential models. In sequential models each batch is fed though the network one after another. Using a recurrent model, for each sample, a time sequence has to be generated first. Since the LSTM is not the first layer and gets feature maps as an input, these maps have to be computed and stacked. For computing feature maps the inference of the preceding layers has to be done.

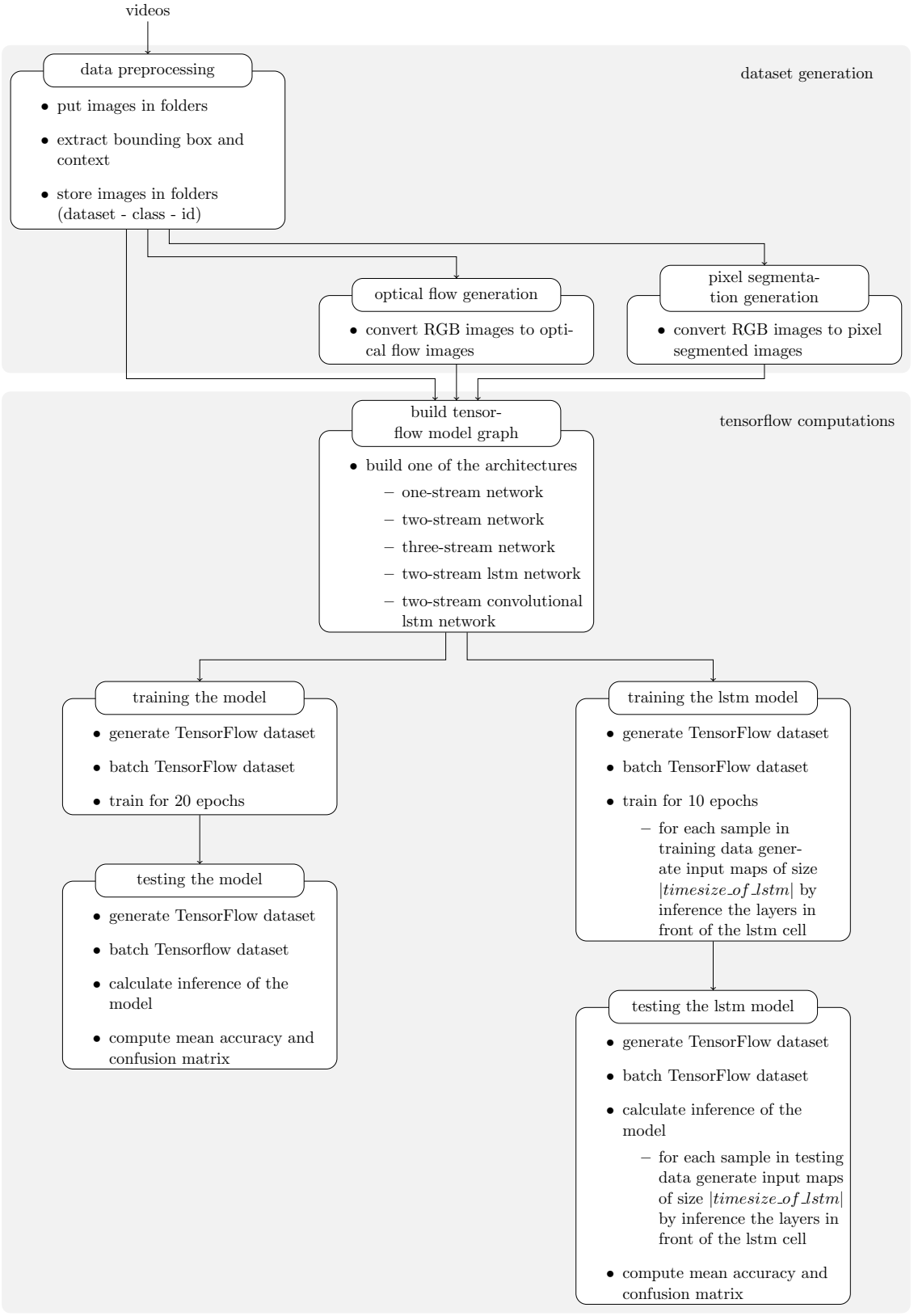


FIGURE 3.1: Workflow of the implementation for this thesis, including dataset preprocessing, model generation, training and testing.

4 Validating the Network Architecture

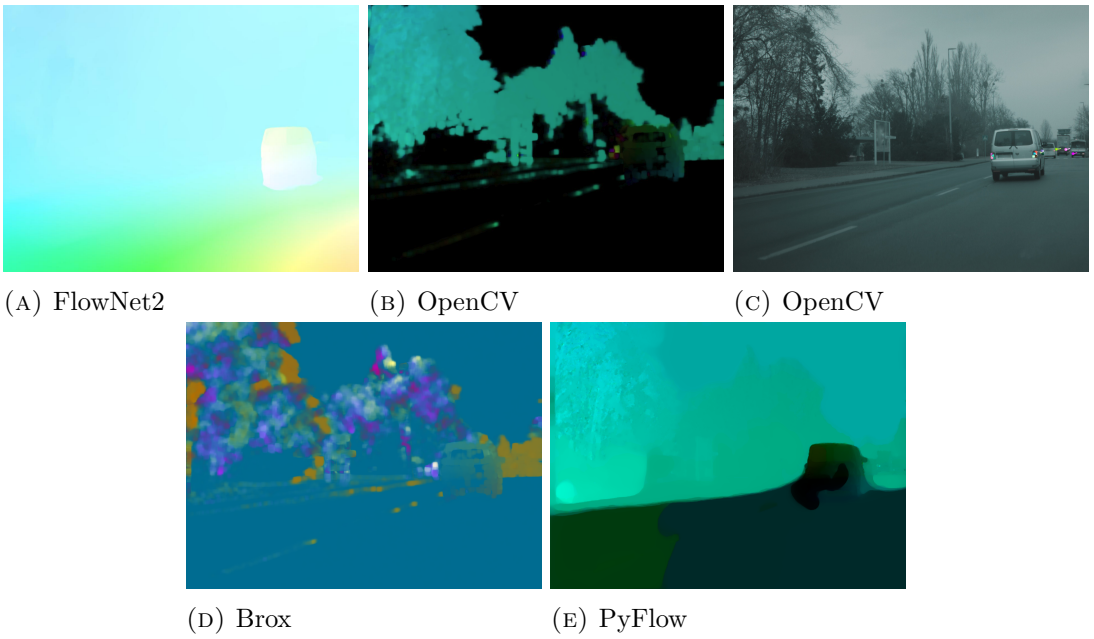
To validate the built network architecture and functionality, the primary aim of this thesis is reaching approximately the same results as presented by Varol et al. [19]. The dataset UCF101 is used to fulfill this task. It consists of 101 everyday activity classes, like *apply makeup, play football* and *yo-yo*. Since they implemented their approach by utilizing Torch, it had to be reimplemented by using TensorFlow for this thesis.

4.1 Dataset Preprocessing

For dataset generation the videos were sampled with 25 frames per second and resized to 171×128 pixels. The input images were then randomly cut into patches with size 112×112. Since the network receives a time sequence as an input rather than single images, the images were stacked to sequences of size 16. This is done by a sliding window approach, with 75% overlap. Missing images at the end of a video were replaced by duplicates of the last existing image.
The presented approach is a two-stream network using RGB images and optical flow as inputs. The optical flow was pre-processed by the FlowNet approach, which will be presented in detail in the section below. The input of the second stream was cropped and stacked by the sliding window approach in a similar fashion.

Optical Flow Preprocessing

Since one of the streams is using optical flow as input, either a network for calculating the optical flow has to be put in front of the stream or it has to be pre-processed. To avoid the extra inference time of the network for each sample, the optical flow was preprocessed for this thesis.
Varol et al. [19] compare three approaches for optical flow generation MPEG flow [59], Farneback [60] and Brox [61]. After analyzing the network accuracy based on the three optical flow outputs, the authors decided to use the Brox approach. Compared to the actual motion in the videos, Brox generates the best results on the UCF101 dataset as well.



(A) FlowNet2 (B) OpenCV (C) OpenCV

(D) Brox (E) PyFlow

FIGURE 4.1: Different optical flow approaches. (A) shows the chosen method used in this thesis.

In this thesis five optical flow approaches were compared. FlowNet2 [57] is a convolutional neural network (CNN) based approach and compared to the actual motion in the videos. It generates the best results on our dataset and is able to handle small displacements as well as large displacements. Next to this, two OpenCV approaches based on Lukas Kanade [56], Brox [61] and PyFlow [62] have been compared, see Figure 4.1. As it can be seen, FlowNet2 is the only approach in which the motion-areas are not highly fragmented. While FlowNet2 and PyFlow are both neural network based approaches, the three remaining methods are based on features and moving pixels. In many frames as well as in Figure 4.1 both approaches, FlowNet2 and PyFlow generate good results. But still FlowNet2 outperforms PyFlow, see Figure 4.2 for another example. While with the optical flow computed by FlowNet2 it is possible to distinguish the motion of each individual, this is not possible in PyFlow.
In summary, it can be said that the FlowNet2 approach produces better results and is therefore chosen for all further steps.

4.2 Network Architecture

As depicted in Figure 4.3, and based on the architecture described by Varol et al. [19], the used network consists of five 3D-convolutional layers, with 64, 128, 256, 256, 256 filter maps followed by three fully-connected layers with the sizes 2048, 2048 and number of output classes. The size of the filters is 3x3x3 for each convolutional layer. Each

(A) original (B) FlowNet2 (C) PyFlow

FIGURE 4.2: Comparison of FlowNet2 and PyFlow.

convolutional layer is followed by a rectified linear unit (ReLu) and a 3D max pooling layer. Each max pooling filter has the size 2x2x2, except for the first which is 2x2x1. By repeating one pixel in all three dimensions the convolutional output is kept constant. The filter stride is one for each convolution and dimension and two for each pooling dimension. In the first two fully-connected layers a dropout layer is used, and each is followed by a ReLu. A Softmax layer outputs the class scores.

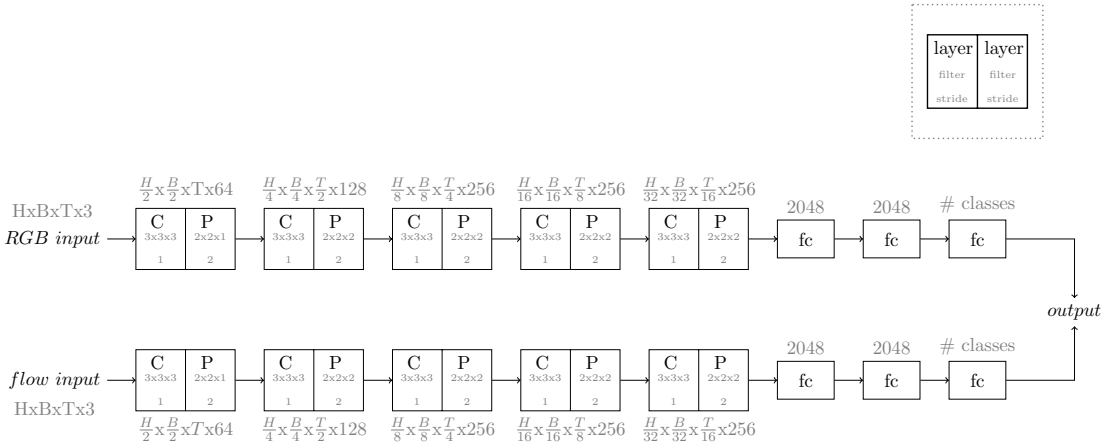


FIGURE 4.3: Network architecture. While C describes a convolutional layer, P describes a pooling and fc a fully-connected layer. The size of filters F is 3x3x3 in each convolutional layer. The convolutional layers are followed by a ReLu as well as a 3D pooling layer. The size of the pooling filters is 2x2x2, except of the first, which is 2x2x1. Filter stride S for all dimensions is 1 for convolution and 2 for pooling.

4.3 Training and Evaluation

Based on the proposed approach of Varol et al. [19] the network was trained for eight epochs, where each epoch consists of 17573 mini-batches of size 23. The initial learning rate was 3×10^{-3} and was decreased twice by the factor of 10^{-1} after 80 thousand and 125 thousand steps. For regularization the dropout was set to 0.5 and weight decay was initialized with 5×10^{-3}. It was reduced equally to the learning rate. Gradient descent with a momentum of 0.9 was used as the optimizer.

The measure of quality used by Varol et al. [19] is based on the mean accuracy per video clip. Based on the calculated accuracy for each input sequence they average all softmax scores for one video and take the maximum value as prediction.

During the implementation process of the network presented in Varol et al [19], different problems arose. In first tests, the distribution of the training dataset prevent the model from generalization. This issue was fixed by shuffling the dataset. The authors specify a mini-batch size of 30, but only 26 mini-batches fit into the used RAM. After training the network, only the reported results of the RGB stream could be approximately reproduced with a difference of 4 percentage points. The accuracy of a combined flow and RGB-stream model was better than a single stream, but compared to the results in the paper, which reports an accuracy of 77% for the flow-stream, the results were much lower and only score an accuracy of up to 51%. Since the paper presents multiple approaches, it was noticed that a seemingly different approach with the mini batch size of 10 got the same results. After retraining the network with the new batch size the accuracy improved by one percent, which is still far beyond the presented results.

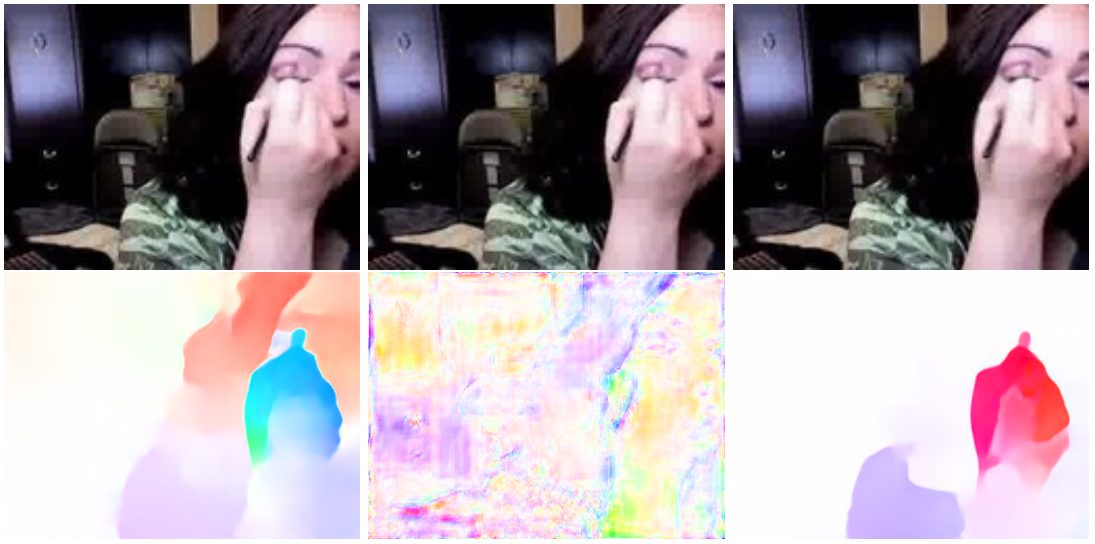


FIGURE 4.4: Input images for the multi-stream network. The first row represents the RGB input and each image in the second row represents the optical flow up to the image which can be seen above. Which means, that the second optical flow image is the result from the first RGB image to the second.

A reason for this might be the generated optical flow, by looking at the dataset it can be seen that in some cases two following frames are similar, this causes a very noisy optical flow image. Since following frames are stacked to sequences, noisy images can prevent the network from recognizing the motion, because some images do not show any meaningful optical flow, see Figure 4.4. The reason for this problem might be based on the image sampling method. To test the network, a second dataset was sampled by a lower frame rate with the intention of lowering the similarity of two consecutive frames.

But even by using a lower frame rate, the problem could not be solved. Another reason for this could be the chosen optical flow algorithm.

Even though this work cannot reproduce the classification accuracy of the paper by using the optical flow stream, this is possible with RGB images. By training and testing the RGB stream network the results presented in the paper were reproduced. Fusing the two-stream to a multi-stream network improved the accuracy by 6 percentage points, which indicates the functionality of the two-stream network as well. All in all, it can be said that by training and evaluating on the UCF101 dataset it was shown that it is possible to fulfill the task of action recognition with this presented approach.

5 Dataset Preprocessing

After providing the functionality of the used network, it is applied to a different dataset. While the UCF dataset contains videos of everyday activities, the new dataset contains tactical driving maneuvers. Most of the activities in the UCF dataset are very different to each other, but many driving maneuvers are very similar. The difference between a `lane changing` vehicle and a `preceding` vehicle is only defined by the passing of the lane markings, but both vehicles are driving in approximately the same direction.
This chapter deals with finding a proper data set as well as a suitable metric.

5.1 Dataset Preprocessing

The maneuver datasets contain grayscale images of size 1280×960. For each image, there is a list of tracked objects, that can be found in this image with its corresponding ID over time. Moreover, it includes classification data for objects and actions as well as the location of bounding boxes.
Due to the fact that each bounding box contains only the object and no surrounding context (e.g., other objects and lane markings that are important for the `lane change`), each object was cut with its context and stored in combination with its ground truth values. Since it is not trivial to decide how much context is needed, different options were tested, see Section 6.2 *Context Size of Tracked Objects*.

The dataset consists of 13 different classes. The distribution of the classes in the dataset can be seen in Figure 5.1. The network will be trained with the balanced dataset, see 5.1 since the network did not converge on the real dataset on first tests. The network takes images of a fixed size, but the bounding boxes have different sizes. For this reason they are normalized to a fixed size of 152×132. To determine this, the distribution of bounding box sizes was examined and a mean value was taken. To add content to the bounding boxes, each box was at least extended to 60% in x and y direction. If the box and its context was still to small to fit the size of 152×132 more context was added if there was spare context remaining. If there was no context left, the last column of the image was padded. If the image was too large to fit the size, it was expanded until

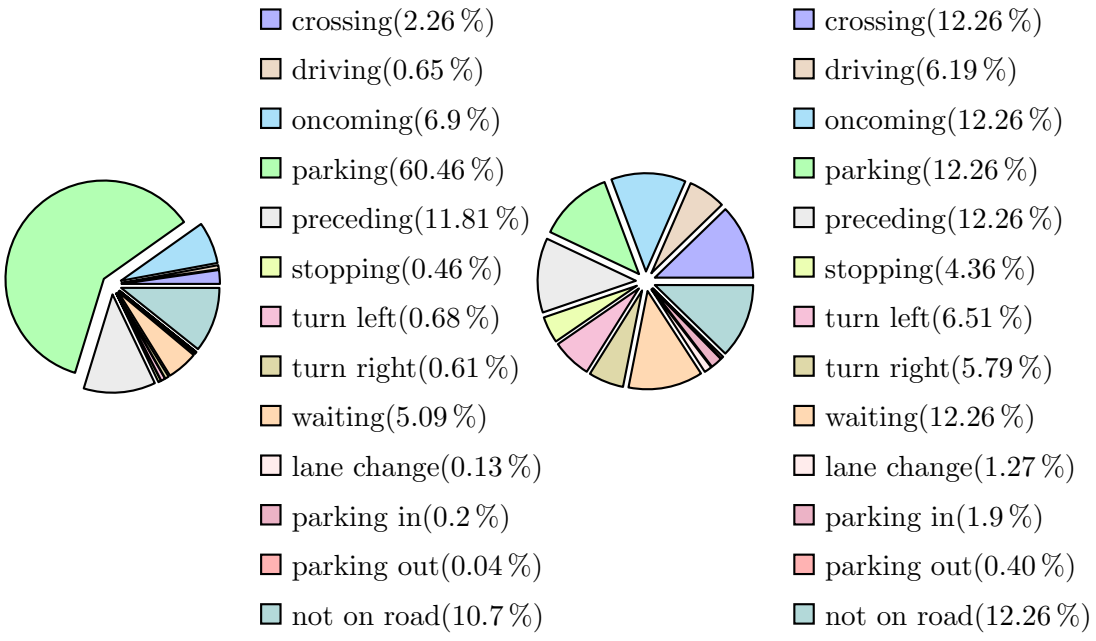


FIGURE 5.1: The training data set before the balancing process can be seen on the left and after balancing on the right.

it had the same x and y proportion and then rescaled to 152×132. This prevents the image from changing perspective.

Pixel Segmentation Preprocessing

For the third stream pixel segmented images have to be generated. Some of the classes can be classified based on the street markings. Since one of the classes in the pixel segmented image is `street marking` this seems to be a useful input. The pixel segmented input images are created by an implementation of the ICNET [63]. The network was trained on automotive video data. Since there is no ground truth data for pixel segmentation of the used dataset in this thesis, a pretrained architecture was used. By looking at the images in Figure 5.2 it can be seen, that some pixel segmentations are very fragmented, the reason for this might be the different dynamics due to differently chosen camera settings compared to the training data. The matching from object to color can be seen in Table 5.1.

It can be seen, that especially in the third row of Figure 5.2 the network could not distinguish between `background` and `non drivable area`. Moreover in the first row the presented vehicle is highly fragmented, but the lane markings have a very good quality. This can be seen in most of the images with lane markings. Zebra crossings as well as signs presented in the third row are often distinguished. Furthermore, in the first row it can be seen that the vehicle's tires are classified as pedestrians.

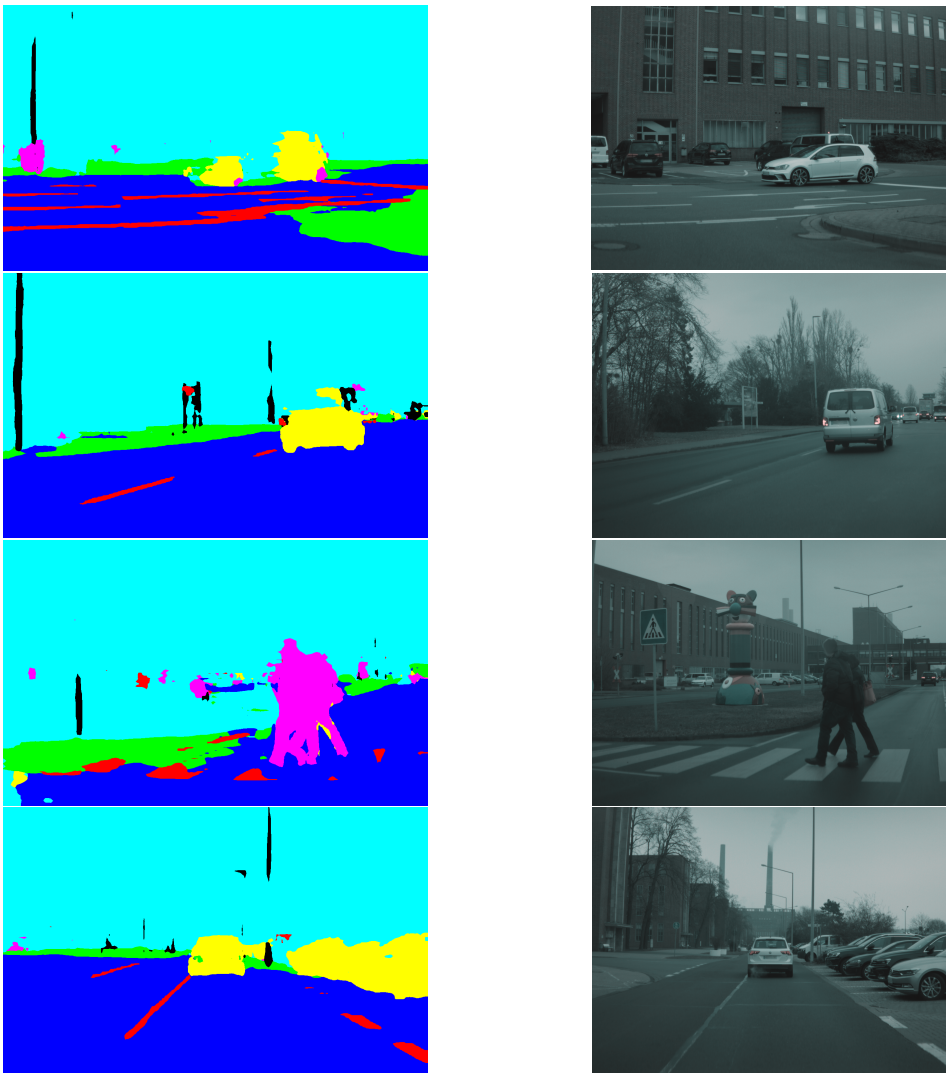


FIGURE 5.2: Pixel-segmented image from ICNET network.

color	class
turquoise	background, obstacle, building
magenta	pedestrian
yellow	vehicle
blue	drivable area
green	non drivable area
red	signs, markings
black	poles, trees

TABLE 5.1: Matching of the colors seen in the pixel-segmented image to the classes.

5.2 Dataset Comparison

The presented dataset in the section above is highly unbalanced and it is obvious, that some of the underrepresented classes are difficult to learn. Since recording and labeling new data needs much time it its not possible to work with bigger datasets during this thesis.
Therefore, publicly available data sets were compared with each other. Table 5.2 displays that there is no possibility to work on one of those, because labeled and tracked actions of objects recorded in urban environments are needed. The `JAAD` dataset is the only one containing labeled actions *and* tracked objects. But the authors record the videos from a predefined place rather than the ego position of a driving vehicle. Furthermore it only consists of pedestrians.
Since none of the publicly available data sets are suitable for the task, it is not possible to consider other datasets at the moment.

dataset	labeled actions	tracked objects	annotations
KITTI [64]	✗	✓	
Cityscapes [65]	✗	✗	
Comma.ai [66]	✗	✗	
RobotCar [67]	✗	✗	
BDDV [68]	✗	✗	
Udacity [69]	✗	✗	
GTA [70]	✗	✗	
JAAD [71]	✓	✓	only classified pedestrians, and not from ego perspective of a driving vehicle
CamVid [72] .	✗	✗	
NTSEL [73]	✗	?	not public, includes four pedestrian activity classes
NDRDB [74]	✗	?	not public, includes four pedestrian activity classes
MOT Challenge [75]	✗	✓	

Table 5.2: Publicly available datasets in the field of highly automated driving.

5.3 Metric

Knowing that the dataset is highly unbalanced, the accuracy over all classes is biased towards the most frequent class. For this reason, in this thesis the accuracy a will be described as:

$$a_c = \frac{true\ positive + true\ negative}{true\ positive + true\ negative + false\ positive + flase\ negative} \tag{5.1}$$

$$a = \frac{\sum a_c}{|c|}. \tag{5.2}$$

Here a_c is the accuracy of one specific class c and $|c|$ represents the total number of classes.

The accuracy in the maneuver dataset evaluation differs from the presented approach by Varol et al. [19], because averaging the softmax output of multiple inferences is not possible in reality. They collect the prediction of samples within the same video and take the mean softmax. The maximum value determines the prediction. Since objects can change their actions this method is not transferable to the task of action recognition in highly automated driving.

6 Examining Architectures for the Task of Action Recognition of Tactical Driving Maneuvers

After finding a proper dataset, this chapter analyzes the possibility of using the network, presented by Varol et al. [19] in the field of maneuver recognition. In the following sections, multiple tests are evaluated regarding different fusion methods as well as different dataset augmentations and input streams.
Since the result of one single test often motivates the next experiment, each section includes the used methodology as well as the execution. After analyzing multiple tests, the results are discussed in a critical reflection in the last section of this chapter.

6.1 Comparing RGB-Stream, Optical Flow-Stream and Multi-Stream Networks

In this section an *RGB*, *optical flow* and a *multi-stream* network will be compared and evaluated. The multi-stream approach combines the RGB and flow stream by averaging their softmax outputs.

Assumption: With the combination of RGB and optical flow, it is possible to improve the classification of maneuvers that are highly dependent on movements.

It can be assumed that using only RGB data biases towards their respective dominantly shown side; vehicles driving in front present their rears while trailing vehicles' fronts are predominantly visible. With the addition of the optical flow the differences between `parking` and `turn left`, `turn right` will be more pronounced in the data than by only using the RGB stream.

Evaluation: The confusion matrices show an improvement of the classification. Since stopping vehicles were often misclassified as `oncoming` or `parking` by using only the RGB stream, the class accuracy increased but overall the mean accuracy still decreased.

This is plausible, since almost all image related features, for example lane markings are lost when generating the optical flow.
Comparing the single-stream networks to the two-stream network, it can be seen that the accuracy increased.

	RGB stream	optical flow stream	multi-stream softmax fusion
mean accuracy	31.4	29.4	35.2

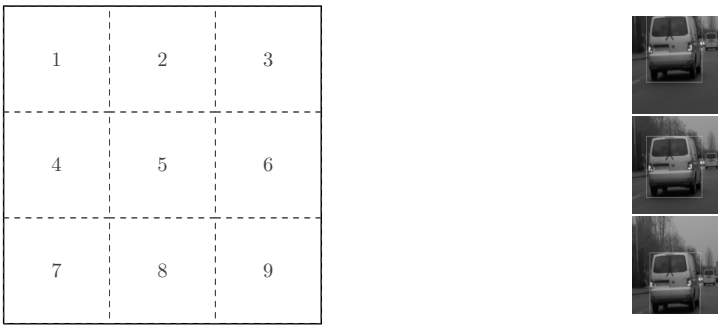
TABLE 6.1: Results using only the RGB stream or the optical flow stream and a fusion architecture of optical flow and RGB by averaging the softmax outputs.

In this test the fusion method was similar to the presented approach of Varol et al. [19]. The softmax output of the multi-stream network is simply the mean softmax of both streams. This clearly has the disadvantage that the individual features of the different streams cannot be combined. Because of this, different fusion methods will be analyzed in the following.
Since the dataset slightly changed for the next sections, the mean accuracy in Table 6.2 is different than in Table 6.1.

6.2 Choosing the Context Size of Tracked Objects and Fusion Methods

Since for each object the image is cut into a new 152×132 image it its important to decide where to place the object. This is no trivial task, because it needs to be considered which part of the context surrounding to the object can be cut and which should be kept. Because of this, nine different options were considered, see Figure 6.1. The three rightmost cases and three leftmost cases can be discarded, because the importance of the context to the right and to the left is highly dependent of the road layout. This means, if there is a right turn, the context to the right is more relevant than it would be for a left turn.
To compare the three remaining possibilities all of them were tested with three different architectures, which are shown in Figure 6.2. The first architecture is described by Varol et al. [19] and is averaging the softmax output of each stream.

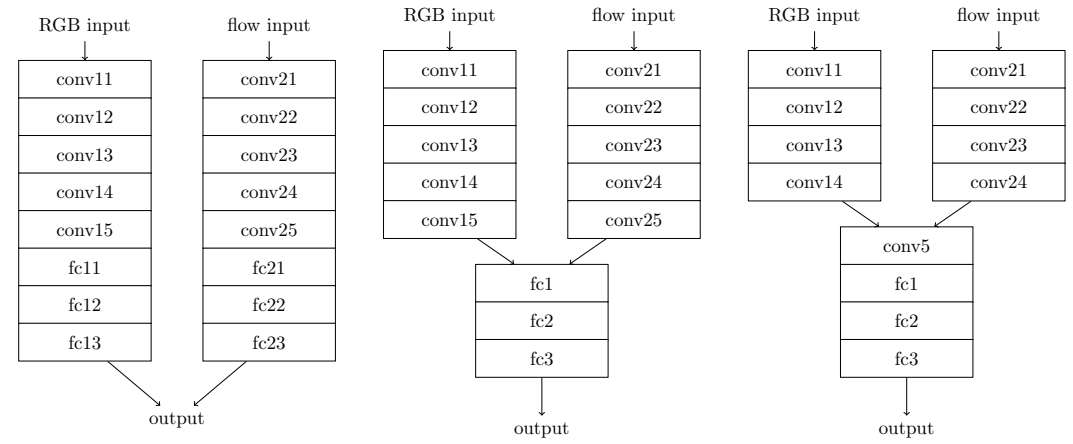
Assumption: By fusing the streams at the softmax layer it is not possible to combine important features of both streams, because they are not disjoint. By fusing earlier, it might be possible to benefit from the different features that are extracted in the two streams and their local correspondence. The second (B) and third (C) architecture in Figure 6.2 show early fusion methods after the 5^{th} and 4^{th} convolutional layer.



(A) (B)

FIGURE 6.1: (A) Nine different locations for object placement in the image. (B) Three example images for object placement 2, 5, and 8.

Furthermore, it is assumed, that adding context in all four dimensions will yield the best classification results. The context of each direction is important for different cases. By looking at e.g., a `lane changing` vehicle, the context below the vehicle (in the image) might be more important than the context above the vehicle (in the image) because the lane markings are visible there. While it is in general not possible to see the lane markings right in front of a `preceding` car, it is necessary to watch them behind a `preceding` car. For `preceding`, `waiting` vehicles, it might be more interesting to know the context above a certain object (in the image), since an obstacle or a traffic light causing the waiting maneuver might be located there.



(A) (B) (C)

FIGURE 6.2: Showing the three different fusion methods. (A) shows a fusion by averaging the softmax outputs of the different streams. Subsequently (B) and (C) show a fusion after the 5^{th} and 4^{th} convolutional layer.

Evaluation: Table 6.2 presents the per-class and mean accuracy of the different tests. By looking at the table it can be seen that the network architecture fused after the 5^{th} convolutional layer performs best. Furthermore, the three different cuttings perform similar for this fusion method.

	top center			center center			bottom center		
	softmax	5th cnn	4th cnn	softmax	5th cnn	4th cnn	softmax	5th cnn	4th cnn
crossing	60.79	37.53	54.56	68.26	**73.03**	63.28	45.33	70.12	72.61
driving	34.65	**39.07**	21.78	26.73	37.62	14.85	18.68	34.16	26.73
oncoming	56.52	**69.83**	57.98	59.66	59.26	58.71	64.01	57.69	55.69
parking	56.43	**68.57**	58.3	59.36	55.77	52.97	47.23	59.67	53.23
preceding	49.6	**69.86**	46.16	53.11	58.15	43.96	61.38	54.12	48.79
stopping	12.33	34.59	27.74	25.68	24.32	30.48	33.13	31.16	**61.38**
turn left	46.61	46.83	30.77	43.44	44.34	41.18	**58.97**	45.70	51.58
turn right	31.25	21.29	31.25	25.69	29.17	27.78	40.64	**42.36**	34.72
waiting	**70.46**	62.05	53.43	54.56	65.81	63.73	58.33	49.21	46.2
lane change	1.04	2.59	4.17	0	2.08	**16.67**	1.04	12.50	4.17
parking in	0	0	0	0	3.33	0	0	**10.00**	5.56
parking out	0	0	0	0	0	0	0	0	0
not on the road	68.27	**89.46**	70.23	71.7	71.67	62.63	87.13	67.07	65.53
mean accuracy	37.53	40.90	35.11	37.55	40.35	36.63	39.69	**41.07**	39.18

TABLE 6.2: Results for the three different object locations, each was tested with three architectures. For each class the best classification result is highlighted in bold. The accuracy for each class is presented in %.

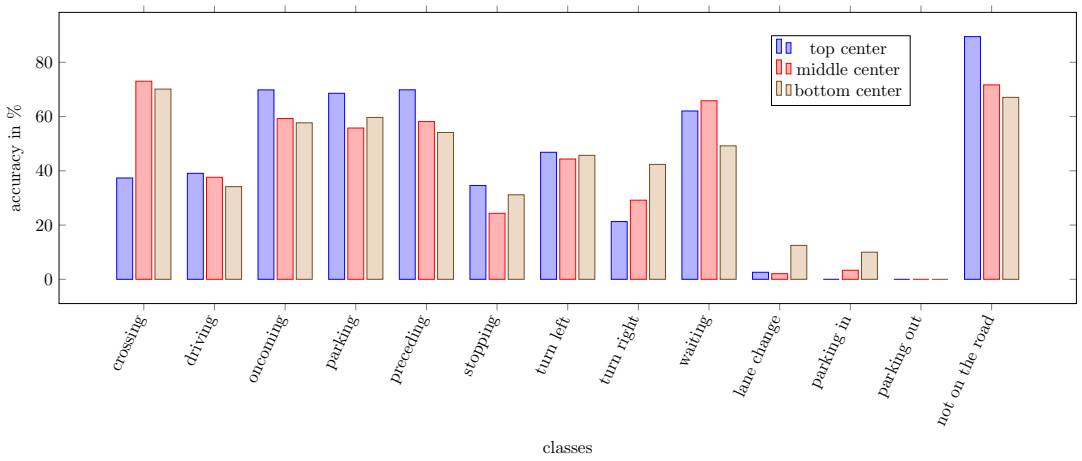
Figure 6.3 (B) shows a plot of the accuracy of the different architectures with top center input cutting. It can be concluded, that the fusion method after the fifth convolutional neural network (CNN) has the highest accuracy in most of the classes. Especially in maneuvers involving quick motions, such as `driving`, `turn right` and `lane change`, it outperforms the others.

Considering the per-class accuracy, it is notable that each of the cuttings have different advantages for different classes. While `lane changes` can be best classified by the bottom center cutting, the top center cutting is able to classify oncoming and preceding vehicles the best, see Figure 6.3 (A).

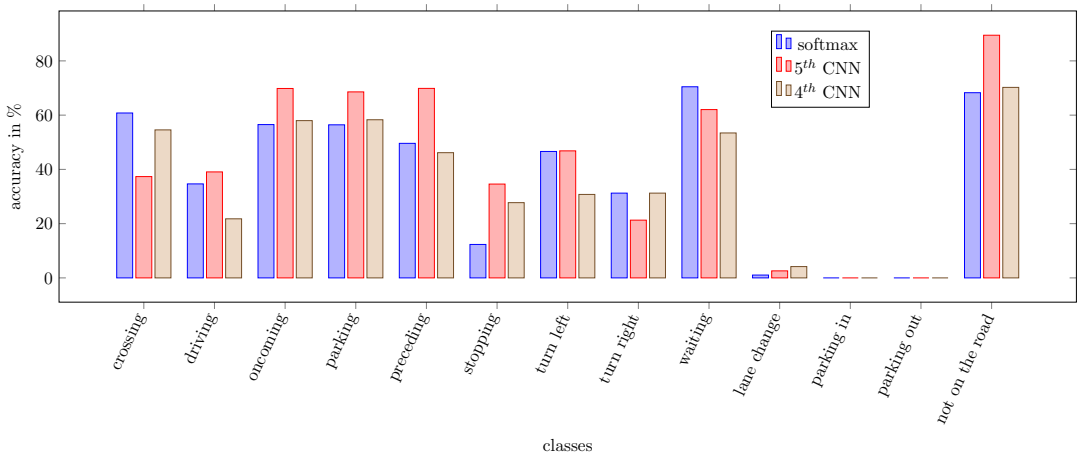
classes	variance
crossing	50.45
driving	10.00
oncoming	24.19
parking	14.04
preceding	38.24
stopping	4.22
turn left	70.17
turn right	0.22
waiting	16.07
lane change	1.16
parking in	0.37
parking out	0
not on the road	10.51
mean accuracy	0.8

TABLE 6.3: The variances of the different classes for the top center crop and the architecture that fuses the streams after the 5th convolutional layer.

The variance of the accuracies of the different classes for the top center crop and the



(A) Showing different context cuttings.



(B) Showing different architectures.

FIGURE 6.3: A plot of the three different cuttings can be seen in (A), each with the best architecture. The three different architectures are plotted in (B), the top center cuttings were used as input.

architecture that fuses the streams after the 5^{th} convolutional layer is depicted in Table 6.3. It can be seen that the variance differs between different classes and have an especially high value for the classes `crossing` (50.45), `preceding` (38.24) and `turn left` (70.17). But the variance for the mean accuracy is very low (0.8). This means that it is not possible to decide which cutting performs best. Since the standard deviation is about 1, all cuttings have approximately the same accuracy.

Concluding, these results suggest that more context is needed for better classification results. Since the aim of this test was to find the best architecture as well as the best context cutting, it is necessary to test at least one more cutting with extended context above and below the object. Because the generation of the new dataset and testing the architecture once again requires time, this test will be done after the parameter optimization in Section 6.6. For the next tests the top center cutting will be used. It

outperforms the others in five classes. This is compared to the other cuttings the best result.

6.3 Optimizing Parameters

To optimize the parameters of the chosen architecture, different parameter settings are tested. First of all, multiple learning rates are tested for different optimizers. After finding a proper learning rate, different combinations of batch sizes and dropouts are evaluated. It is worth mentioning at this point, that the parameter optimization is not the main focus of this thesis.

To avoid testing different parameters manually, tuning algorithms can be found in literature. Baydin et al. [76] introduce a method for online learning rate adaption, since the learning rate is "often the single most important hyperparameter" [76]. With the use of this approach it is possible to optimize "the loss trajectory closer to the optimal one" [76]. This is done by applying gradient decent on the learning rate and is based on the partial derivative of an objective function. The following update step is done with respect to (w.r.t.) the learning rate as follows:

$$\alpha_t = \alpha_{t-1} - \beta \frac{\delta(\theta_{t-1})}{\delta\alpha} \tag{6.1}$$

$$\theta_t = \theta_{t-1} - \alpha_t \nabla f(\theta_{t-1}), \tag{6.2}$$

with the learning rate α, the objective function θ and the hypergradient learning rate β.

By looking at the results of Baydin et al., it can be seen that with the use of this algorithm it is possible to improve the results of the Adam and Momentum optimizers without a time intensive tuning of the learning rate. While the authors assume, that the loss of two batches is almost the same this approach might only work with large batch sizes. The tested learning rate decay methods are called *two times* $\times 10^{-1}$, which is equal to the presented method by Varol et al. [19]. This means, the learning rate is multiplied by 10^{-1} two times during the training process. Next to this, experiments *without learning rate decay* were tested as well as an *exponential* approach, in which the learning rate was multiplied with 0.9 after every 4000 steps.

Assumption: It is possible to increase the mean accuracy by optimizing the parameter setting of the network. As mentioned in [76], one of the most important parameters is the learning rate, which is the reason for optimizing it first.

Evaluation: The results, are compared to the mean accuracy of the architecture, which fuses the RGB and optical flow stream after the 5^{th} convolutional layer. The tests are based on the dataset, including the object in the top center of the image.

Multiple variations of learning rates and optimizers have been tested. Table 6.4 shows that the presented results are very similar to each other. It is not possible to choose a single best parameter setting which is significantly outperforming the other settings. Since an amount of 18 tests for finding a proper learning rate is not representative we can assume that the results are only local optima and by performing a more through parameter search it should be possible to find a better result. Next to this it can be seen, that the presented hyper parameter decay from Baydin et al. in [76] performs worse than the other tests. This might be because of the small batch size of only 10 samples.

optimizer	initial learning rate	learning rate decay	mean accuracy
Momentum	0.003	two times $\times 10^{-1}$	40.9
Adagrad	0.003	two times $\times 10^{-1}$	44.38
Adagrad	0.003	no decay	45.8
Adagrad	0.01	no decay	44.01
Adagrad	0.003	exponential	45.12
Momentum	**0.0001**	**exponential**	**46.35**
Adam	0.003	hyperparameter decay	41.78

TABLE 6.4: Comparing different learning rates and optimizers. The first row represents the baseline. For each learning rate and decay the optimizers Adam, Adagrad and Momentum were tested, but only the best result is represented in this table. Each parameter setting was trained for 20 epochs.

Besides finding a proper learning rate, some of the best results from Table 6.4 were tested with a higher dropout of 0.9, which means that one node is kept with a probability of 0.9 and a lower dropout of 0.3. The test results are presented in Table 6.5. Similar to Table 6.4, no result is outstanding, but it was possible to increase the mean accuracy by 2.7%. Compared to the baseline architecture this leads to an improvement of 7.2 percentage points.

A comparison of the per-class results of the best model after optimizing the parameters, with the baseline model is depicted in Table 6.6. It can be seen that the accuracy increased for many classes. One outstanding result can be seen in the class `not on the road`. It is now possible to classify over 90% of pedestrian, `standing on the road`. But there are still problems in many of the remaining classes. By looking at the class `crossing` it can be seen, that the network can correctly classify only 55% of the samples. Although both classes are equally represented in the balanced dataset the results differ a lot. Comparing classes that are better represented to classes with fewer samples, it can be seen that highly represented classes have in general a higher accuracy. This

optimizer	initial learning rate	learning rate decay	dropout	mean accuracy
Momentum	0.0001	exponential	0.5	46.35
Adagrad	0.01	no decay	0.9	38.82
Adagrad	**0.003**	**exponential**	**0.9**	**49.06**
Momentum	0.0001	exponential	0.9	46.44
Adam	0.003	hyperparameter decay	0.9	38.82
Adagrad	0.01	no decay	0.3	42.27
Adagrad	0.003	exponential	0.3	44.91
Momentum	0.0001	exponential	0.3	43.40
Adam	0.003	hyperparameter decay	0.3	10.33

TABLE 6.5: Comparing different dropouts. The first row represents the baseline. Each parameter setting was trained for 20 epochs. This comparison is based on the best results of Table 6.4.

can be explained with the big amount of available data. It can be assumed that the classification accuracy can be increased by a larger data set.

By looking at the variances of the optimized models in Table 6.7, it can be seen that the variances are different for each class. They are very small for classes like `parking` and `not on the road` and zero for the class `parking out` since there are no correct classifications at all. The variance for the class `parking in` is around 96 which is a standard deviation of almost 10. The accuracy in this class is between 26 up to 48 with a mean value of 34. Even though this class is highly underrepresented it is possible to reach an accuracy of almost 50 percent in some tests.

All in all, it can be seen that many of the tests have almost the same mean accuracy. Since there is no parameter setting which outperforms the others, it is difficult to decide

	baseline [%]	after optimization [%]	data set distribution [%]
crossing	37.53	55.40	12.26
driving	39.07	52.17	6.19
oncoming	69.83	77.80	12.26
parking	68.57	72.53	12.26
preceding	69.86	65.24	12.26
stopping	34.59	34.19	4.36
turn left	46.83	38.27	6.51
turn right	21.29	26.4	5.79
waiting	62.05	82.37	12.26
lane change	2.59	10.96	1.27
parking in	0	29.27	1.9
parking out	0	0	0.4
not on the road	89.46	93.10	12.26
mean accuracy	40.9	49.06	

TABLE 6.6: Comparing the accuracy per class, with the baseline model and the best model after optimizing the parameters.

classes	variance
crossing	14.80
driving	9.45
oncoming	0.45
parking	0.079
preceding	0.48
stopping	0.16
turn left	10.50
turn right	0.56
waiting	4.97
lane change	1.66
parking in	96.50
parking out	0
not on the road	0.62
mean accuracy	0.71

TABLE 6.7: The variances of the different classes for the model after optimization.

which setting will be used for the following tests. The following tests are based on the best three results.

6.4 Comparing Different Input Images and Bounding Boxes

At the beginning of this chapter, the assumption was stated that input images with drawn bounding boxes around the classified object will output better results than images without bounding boxes. This assumption was shown to be true, because there are many instances in the dataset, in which pedestrians stand right before vehicles or are standing in groups. To distinguish between the different classes and object instances, the bounding boxes should help. Although the object to be classified, is put in the same position in the image, it is still possible that there are multiple objects at this position or in the vicinity.

Assumption: Bounding boxes drawn around objects to be classified increase the classification performance. The reason for this is that distinguishing multiple objects in the same spatial area in the image is hard otherwise.

Evaluation: In Table 6.8 the results of three different tests can be seen. The first approach is based on RGB input images with bounding boxes drawn around the object and precomputed optical flow based on these images. Secondly, it was tested whether using the precomputed optical flow input, based on images without bounding boxes produces better results. Additionally, in a third test it was examined if input images without bounding boxes, decrease the mean average of the network.

	with bbox [%]	only rgb bbox[%]	without bbox [%]
crossing	55.40	53.11	54.98
driving	52.17	28.71	13.86
oncoming	77.80	62.68	63.05
parking	72.53	59.2	63.13
preceding	65.24	59.45	60.38
stopping	34.19	33.90	26.37
turn left	38.27	40.72	39.37
turn right	**26.4**	**36.81**	**43.56**
waiting	82.37	59.33	52.8
lane change	**10.96**	**23.96**	**29.17**
parking in	29.27	3.33	3.33
parking out	**0**	**18.75**	**12.5**
not on the road	93.10	73.7	62.03
mean accuracy	49.06	42.59	40.31

TABLE 6.8: Comparing different bounding box (bbox) options.

It can be seen, that drawn bounding boxes increase the mean accuracy by 7% compared to the test without any bounding boxes. By looking at the confusion matrices of the different tests it is conspicuous that `not on the road - pedestrians` are often swapped with `parking - vehicle`. This observation leads to our assumption that pedestrians standing in front of vehicles are hard to classify, if there are no bounding boxes drawn in the image. Table 6.9 shows the impact of bounding boxes using the example of `pedestrians`. The last row shows right classifications, while the remaining rows show wrong classifications. It can be seen, that the misclassification of `not on the road` into the class `parking` decreased by 16% after drawing bounding boxes in the input image. Furthermore, the true classifications increased by 30%.

	with bbox [%]	only flow bbox [%]	without bbox [%]
crossing	6.67	7.63	11.83
driving	0	1.33	2.1
oncoming	0	0.9	1.03
parking	**0.47**	**9.3**	**16.6**
preceding	0.37	2.5	2.36
stopping	0	0	0
turn left	0	1.76	0.86
turn right	0	0.97	0
waiting	0.13	1.6	1.47
lane change	0	0	0
parking in	0	0	0.23
parking out	0	0	0
not on the road	**92.26**	**73.7**	**62.03**

TABLE 6.9: Comparing confusion matrices of different bounding box options for the class `not on the road - pedestrian`. The last row shows right classifications, while the remaining rows show wrong classifications.

Moreover, it can be seen that the classification for action related classes e.g., `turn right`, `lane change` and `parking out` perform better, without drawn bounding boxes. The reason for this might be the modified optical flow if the computation is based on bounding box images. Since especially the classes `lane change` and `parking out` are highly underrepresented in the dataset, only a few samples can have a significant impact on the accuracy. For this reason, the improvement of the classification result is not as representative as an improvement e.g., of the class `crossing`.

In summary it can be said that the input images used in the previous sections have the best mean accuracy for the tested architecture and parameters. For this reason the input pipeline for the next tests will stay the same and any further computations are based on images with drawn bounding boxes.

6.5 Using Pixelwise Segmented Images for the Third Input Stream

Besides testing different context sizes and image augmentations adding a third stream could increase the classification results. Varol et al. [19] use intelligent dense trajectories as an input for the third stream. With this addition they are able to distinguish between relative motion and absolute motion. They are able to increase the classification results by adding this stream and do not change any parameter setting such as the learning rate or the optimizer.
This thesis is based on tactical maneuvers which are dependent on road markings, streets and surrounding objects. For this reason the input into the third stream consists of pixel segmented images, see Figure 6.4. As presented in Section 5.1 the images are generated by a neural network. The labeled images classify the labels `drivable area`, `road marking`, `vehicle` and `pedestrian`. The stream will be added into the best architecture which fuses the streams after the 5^{th} convolutional layer. Because of the concatenation of the tensors from the three streams, they do not fit on the GPU anymore. For this reason, a lower batch size was used.

Assumption: With the addition of a third input stream it is possible to increase the quality of the classification results. By using labeled pixels as an input it should be possible to distinguish between vehicle and pedestrian related maneuvers. The pre-classified objects help the network to decode the scenes. Next to this it might improve the classification for `lane change`, because this maneuver is highly dependent on the lane markings. Furthermore, the mean accuracy should increase.

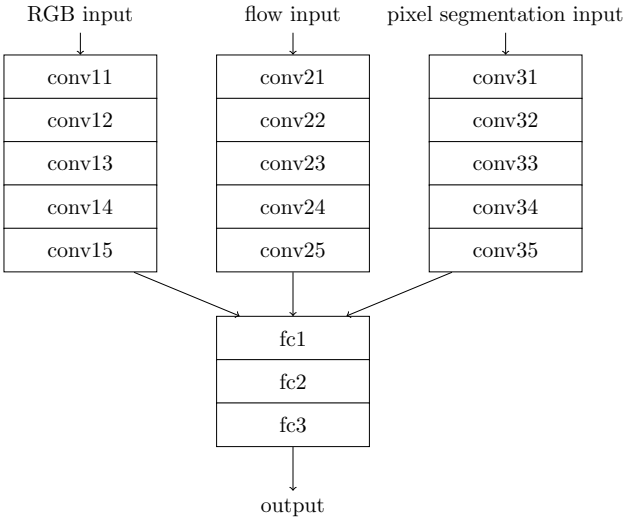


FIGURE 6.4: Architecture of the three stream network with pixel segmented images as input for the third stream.

Evaluation: The three best performing parameter settings were tested on the dataset. All of them perform worse than their two stream counterpart. By investigating the confusion matrix and accuracies per class there is no improvement in any of the classes. The mean accuracies are around 35% which is almost 10 percent points worse than the two-stream network, see Table 6.10. The explanation of this is difficult, since many issues can lead to this result. The number of training epochs were increased from 20 up to 50 but there was no noteworthy improvement visible. One cause for this might be the lowered batch size.

parameter setting				three-stream	two-stream
optimizer	leraning rate	decay	dropout	accuracy [%]	accuracy [%]
Adagrad	0.003	exponential	0.9	35.64	49.06
Momentum	0.0001	exponential	0.9	35.74	46.44
Momentum	0.0001	exponential	0.5	35.14	46.35

TABLE 6.10: Comparing three different parameter settings for the three-stream network.

By looking at Figure 5.2 it can be seen that the precomputed pixel labeling is far from ground truth quality and the objects are highly fragmented. Next to the fragmentation, there is a high amount of false classifications, see the first image for example. In this image the ties are classified as pedestrians.
Besides looking at the input images, a new optimization of the architecture and parameter setting might be useful. Although Varol et al. [19] did not optimize their parameters again, in general each new architecture should be optimized. Parameters cannot be used for different architectures in an universal manner.

6.6 Testing an Extended Context Cutting

In Section 6.2 different context cuttings were tested and discussed. It could be seen that adding context to the top leads to improvements in different classes than adding context to the bottom of the object. Because of this it was decided to do an extra cutting with more context to the bottom and to the top, to evaluate if this improves the classification results. Since generating the new dataset and training the network is time-consuming, the parameter optimization was done in between. The results are compared to the optimized model in Section 6.3.

Assumption: By adding extra context to both, the top as well the bottom of the object, it is possible to increase the accuracy. The baseline from Section 6.3 takes input images with only context added to the bottom. By looking at the results from Section 6.3, it can be assumed that it is possible to increase the accuracy for the classes `crossing`, `turn right` and `lane change` as well as the mean accuracy. The mean accuracy should increase since there is more information available in the image, because of the extended context.

	baseline [%]	extended context [%]
crossing	**55.40**	**51.89**
driving	52.17	42.49
oncoming	77.80	75.91
parking	72.53	77.46
preceding	65.24	79.78
stopping	34.19	28.30
turn left	38.27	54.23
turn right	**26.4**	**20.53**
waiting	82.37	76.53
lane change	**10.96**	**18.10**
parking in	29.27	0
parking out	0	0
not on the road	93.10	91.95
mean accuracy	49.06	47.48

TABLE 6.11: Comparing the accuracy per class, with the baseline model from Section 6.3 and an extended context input.

Evaluation: By looking at Table 6.11 it can be seen that the mean accuracy did not increase. It is hard to find reasons for this. The accuracy not only decreases in the sparsely represented classes in the dataset, such as `parking in` and `parking out` but as well in the highly represented classes `crossing` and `oncoming`. The accuracy increased not as assumed in `crossing` and `turn right`. The only improvement of the accuracy from the assumption can be seen in the class `lane change`. Next to this, improvements can be seen in `parking`, `turn left` and `preceding`.

In summary, it can be concluded that both context cuttings have advantages as well as disadvantages in some classes and the mean accuracy is almost equal. For this reason the inputs are not changed for the next tests, but it should be kept in mind that there are different well suited possibilities for the input available. Comparing the results to the different cuttings from Section 6.2, it can be seen that the accuracies for the classes `crossing` and `turn right` could still not be reached.

6.7 Examining LSTM for Extracting Long Term Motions

A major drawback of 3D-convolutional networks is the classification of long maneuvers. These networks do convolutional operations in 3D space, which means in spatial and temporal dimensions. If a maneuver lasts longer than one input sequence it is hard to classify, because it cannot be captured by the convolutions. In the field of action recognition for tactical driving maneuvers actions like `lane change` and `turn left` span a longer time interval, compared to `driving` and `walking`.

Lu et al. [24] for example combine a 3D-convolutional network with an convolutional long-short term memory (LSTM) cell. Their work is based on a two-stream network with RGB and optical flow as inputs. The base architecture consists of five convolutional layers followed by fully-connected layers. They replace the fully-connected layers that encode the convolutional features by the LSTM cell. Through the cut of the fully-connected layer the spatial context is still visible. To preserve this, they use a convolutional LSTM instead of a classic LSTM. Using a classic LSTM the inputs would have been flattened resulting in loosing spatial context of the extracted features.

In this section the results of different LSTM approaches will be compared. This means convolutional LSTM (Figure 6.5 right) as well as classic LSTM (Figure 6.5 left) approaches are evaluated. Since the convolutional LSTM outputs a multi dimensional output, a fully-connected layer follows the LSTM. With the help of this layer the computed feature maps can be classified. Next to this, time sequence inputs of length 5 and 10 will be tested.

To generate the input sequence, a sliding window approach is used with an overlap of 11 frames. Each sequence consists of five to ten samples. Since the output of the last convolutional layer is the input to the LSTM layer it is necessary to infer this part of the network to generate the input sequence for the LSTM layer. This means, for each sample in the dataset a sequence of length five to ten is generated by inferring the prefix network (convolutional layers before the LSTM cell). Lu et al. [24] train the LSTM separately.

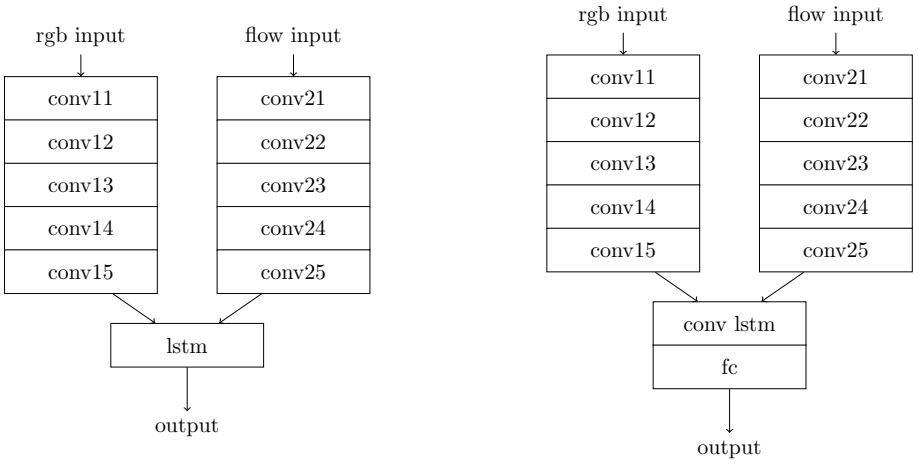


FIGURE 6.5: The architecture with a classic LSTM layer on the left and with a convolutional LSTM layer on the right.

For this reason, the pre trained *prefix network* is used to generate the sequence and only the suffix network (LSTM layer and fully-connected layer) is trained.

Assumption: It is possible to increase the classification result of the network by adding LSTM cells. Further, it is assumed that the accuracy increases especially in classes like `lane change`, `parking in` and `turn left`.

Evaluation: First tests were run with one LSTM layer, a sequence length of 5 and 10 and a batch size of 6 and 10. But none of the models did converge. The next experiments are based on 5 LSTM layers.

To test whether a longer or smaller time sequence is performing better, firstly input sequences of length 5 and length 10 were compared. Because of the long run time of LSTM cells, the two approaches were compared after 5 epochs. The tests were run with a batch size of 10 and 6. But the tests with batch size of 10 did not converge. For batch size 6, it can be seen that an input sequence of length 10 outperforms the smaller sequence. The input of 10 sequences performed twice as good as the 5 time steps sequence input.

Classic LSTMs as well as convolutional LSTMs were tested with a batch size of 6 and a time sequence of 10. Even with trying different parameter settings, the classic LSTM did not converge. Using the convolutional LSTM, multiple parameter settings converged, but overfitted after 5 epochs of training. For the evaluation of the different learning rates, dropouts and trainable variables were varied, see Table 6.12. In the case that only the LSTM and fully-connected layers are optimized, the previously best convolutional neural network (CNN) parameters are used.

lr	lr decay	trainable variables	dropout	result	accuracy
1e-4	high exponential	LSTM + fc	no	no convergence	-
1e-4	low exponential	LSTM + fc	no	no convergence	-
3e-3	high exponential	LSTM + fc	no	convergence	25.74
3e-3	low exponential	LSTM + fc	no	convergence	20.28
3e-3	low exponential	LSTM + fc	yes	convergence	20.22
3e-3	low exponential	whole network	no	convergence	23.34

TABLE 6.12: Different parameter settings for LSTM with different learning rates (lr). Next to this it was examined whether a dropout for the LSM changes the result or not. High exponential learning rate decay means a more frequently decay than with a low exponential learning rate decay.

Comparing the different outcomes in Table 6.12 the results are worse than the baseline without the LSTM layers. Since many of the experiments did not converge there are only a few results to compare with. The results reveals that there is a slight increase of the mean accuracy with a higher exponential decay and by optimizing the whole network.

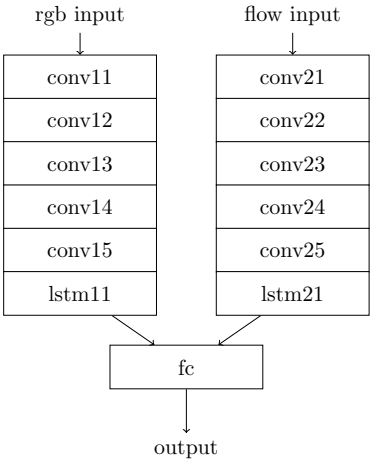


FIGURE 6.6: This architecture uses LSTM layers before fusing the streams.

In a next test it was examined whether the network performs better, if the LSTM cells are put right before the fusion of the different streams, see Figure 6.6. But it was not possible to outperform the model described above. Different parameter setting were tested, but performed worse than the model in Table 6.12. The results can be seen in Table 6.13.

lr	lr decay	trainable variables	dropout	result	accuracy
3e-3	low exponential	LSTM + fc	no	convergence	15.85
3e-3	low exponential	LSTM + fc	yes	convergence	17.73

TABLE 6.13: Two different parameter settings tested for the model depicted in Figure 6.6.

Summarizing this approach, it can be said that it was not possible to increase the classification results by adding an LSTM layer to the baseline network. Reasons for this are hard to locate. It was hard to find a parameter setting that converges by using an LSTM. Through the addition of the LSTM the parameter dimensions grew. The size of the input sequence, as well as the size of the layers, the type of the LSTM cell and their position needs to be considered. It is however possible that there is a suitable parameter setting for this task that was not evaluated in this thesis.

6.8 Critical Reflection

Multiple tests concerning the different streams, their fusion method and the inputs were examined. Looking at the different context sizes of the different fusion methods it can be seen, that one approach was outstanding. The fusion method after the 5^{th} convolutional layer performed best for the three tested input images. But finding a suitable context size for the input image is difficult. All tests yielded similar results and the variances for some classes are very high. It seems like the classification has different strength and weaknesses for different input images, but the mean accuracy is approximately equal. Even after optimizing the parameters, the model performed worse than the baseline in some classes.

By incorporating the bounding boxes in the input images, it was possible to increase the accuracy with marked objects in the input image by about 9 percentage points. But there are still classes that perform better without drawn bounding boxes in the image. By optimizing the parameters the results could be further improved by 8.83 percentage points. This indicates the potential of parameter optimization for the network. Looking at the results, it can be seen that the accuracy increased especially for the highly represented classes in the dataset.

Furthermore the effect of a third input stream was evaluated. The assumption was to improve the classification quality with the additional information. But the results are worse compared to the two-stream network. Reasons for this could be the fragmented input images as well as the smaller batch size.

In the last section, the influence of long-short term memory (LSTM) layers was analyzed. Training the LSTM turned out to be difficult, because many of the tested parameter settings did not converge. The converging models performed worse than the baseline. Finding reasons for this is hard. The parameter dimensions increased with the addition of the LSTM layer. It is possible that there is a suitable parameter setting for this task that was not found yet.

The dataset is highly unbalanced and because of the balancing process many samples of the highly represented classes could not be used for training. For highly represented classes that have a part of 12% (per class) on the dataset, the mean accuracy is almost 74%. Looking at the underrepresented classes that have a part of $0.4-6.51\%$ (per class), the mean accuracy reaches only 27%. Because of this, it can be assumed that adding more data to the dataset can improve the classification results.

7 Summary and Future Work

The aim of this work was to determine whether it is possible to classify maneuvers of surrounding objects and which architecture works the best. Looking at the results depicted in the previous chapters it can be seen, that it is possible to fulfill this task. A mean accuracy of 74% for highly represented classes has been achieved in the experiments. The performance was affected because some classes such as `parking in` were underrepresented. It should be possible to mitigate this and hence increase accuracy by adding new training data. Furthermore, it is shown that the choice of the context in the image influences the accuracies of certain classes. Even after optimizing the model, there were still strengths and weaknesses of different context cuttings visible.

The classification results indicate potential for higher accuracies. Initially, using an LSTM as well as pixel-segmented input images seemed promising, but could not increase the accuracy. The integration of LSTM architectures might be worth exploring. Also further optimization of the parameters (of the model) might grant better results.
Next to this, a combination of different input images should improve the results as well. Since the result of testing different context cuttings have strengths in different classes, it indicates that a combination of multiple cuttings improve the result.

Considering the work of the baseline paper [19], the authors were able to improve their results by expanding the input sequences to up to 100 frames. Since graphics processing units (GPUs) lack sufficient memory capabilities for the expanded sequences it is necessary to further invest in the implementation of the training method. Another option is keeping the sequence length but changing the network model. Further, results shown by Tran et al. [18] indicate an even higher accuracy on the action recognition dataset UCF101 even by utilizing short sequences.

Alongside changing parameters and input streams, other approaches should be considered in future work. Since literature provides different models and approaches for action recognition but almost none for maneuver recognition it is hard to predict which models are useful.

List of Tables

List of Figures

Bibliography

[1] Deutscher Verkehrssicherheitsrat. Schriftreihe Verkehrssicherheit - Vision Zero. Technical Report 16, Deutscher Verkehrssicherheitsrat, 2012.

[2] Dr. Walter Eichendorf and Deutsche Gesetzliche Unfallversicherung. Vision Zero.

[3] Margie Peden, Richard Scurfield, David Sleet, Dinesh Mohan, Adnan Hyder, Eva Jarawan, and Colin Mathers. WHO — World report on road traffic injury prevention. http://www.who.int/violence_injury_prevention/publications/road_traffic/world_report/en/ last access 14.08.2018 14:20, 2004.

[4] VDI Wissensforum. The deployment of Advanced Driver Assistance Systems in Europe. Technical report, VDI Wissensforum.

[5] Statistisches Bundesamt. Verkehr auf einen Blick. page 60, 2013.

[6] Martin Treiber and Dirk Helbing. Microsimulations of Freeway Traffic Including Control Measures. *arXiv:cond-mat/0210096*, October 2002.

[7] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, November 2004.

[8] N. Dalal and B. Triggs. Histograms of Oriented Gradients for Human Detection. volume 1, pages 886–893. IEEE, 2005.

[9] Yan Song, Peiseng Wang, Xinhai Hong, and Ian McLoughlin. Fisher vector based CNN architecture for image classification. pages 565–569. IEEE, September 2017.

[10] Yajing Guo, Xiaoqiang Guo, Zhuqing Jiang, Aidong Men, and Yun Zhou. Real-time object detection by a multi-feature fully convolutional network. pages 670–674. IEEE, September 2017.

[11] P. Wang, L. Liu, C. Shen, Z. Huang, A. v d Hengel, and H. T. Shen. Multi-attention Network for One Shot Learning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6212–6220, July 2017.

[12] Li Zhang, Tao Xiang, and Shaogang Gong. Learning a Deep Embedding Model for Zero-Shot Learning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3010–3019, Honolulu, HI, July 2017. IEEE.

[13] L. Zhu, Y. Chen, P. Ghamisi, and J. A. Benediktsson. Generative Adversarial Networks for Hyperspectral Image Classification. *IEEE Transactions on Geoscience and Remote Sensing*, pages 1–18, 2018.

[14] Georgia Gkioxari, Ross Girshick, and Jitendra Malik. Contextual Action Recognition with R*CNN. *arXiv:1505.01197 [cs]*, May 2015.

[15] Zhichen Zhao, Huimin Ma, and Shaodi You. Single Image Action Recognition Using Semantic Body Part Actions. pages 3411–3419. IEEE, October 2017.

[16] Khaled Saleh, Mohammed Hossny, and Saeid Nahavandi. Early intent prediction of vulnerable road users from visual attributes using multi-task learning network. pages 3367–3372. IEEE, October 2017.

[17] Maryam Asadi-Aghbolaghi, Albert Clapes, Marco Bellantonio, Hugo Jair Escalante, Victor Ponce-Lopez, Xavier Baro, Isabelle Guyon, Shohreh Kasaei, and Sergio Escalera. A Survey on Deep Learning Based Approaches for Action and Gesture Recognition in Image Sequences. pages 476–483. IEEE, May 2017.

[18] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning Spatiotemporal Features with 3D Convolutional Networks. *arXiv:1412.0767 [cs]*, December 2014.

[19] Gül Varol, Ivan Laptev, and Cordelia Schmid. Long-term Temporal Convolutions for Action Recognition. *arXiv:1604.04494 [cs]*, April 2016.

[20] Jinliang Zang, Le Wang, Ziyi Liu, Qilin Zhang, Zhenxing Niu, Gang Hua, and Nanning Zheng. Attention-based Temporal Weighted Convolutional Neural Network for Action Recognition. *arXiv:1803.07179 [cs]*, March 2018.

[21] David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Graham W. Taylor, Rob Fergus, Yann LeCun, and Christoph Bregler. Convolutional Learning of Spatio-temporal Features. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision – ECCV 2010*, volume 6316, pages 140–153. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[22] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Convolutional Two-Stream Network Fusion for Video Action Recognition. *arXiv:1604.06573 [cs]*, April 2016.

[23] Karen Simonyan and Andrew Zisserman. Two-Stream Convolutional Networks for Action Recognition in Videos. page 9, 2014.

[24] N. Lu, Y. Wu, L. Feng, and J. Song. Deep Learning for Fall Detection: 3D-CNN Combined with LSTM on Video Kinematic Data. *IEEE Journal of Biomedical and Health Informatics*, pages 1–1, 2018.

[25] Mohammadreza Zolfaghari, Gabriel L. Oliveira, Nima Sedaghat, and Thomas Brox. Chained Multi-stream Networks Exploiting Pose, Motion, and Appearance for Action Classification and Detection. pages 2923–2932. IEEE, October 2017.

[26] Zhigang Tu, Wei Xie, Justin Dauwels, Baoxin Li, and Junsong Yuan. Semantic Cues Enhanced Multi-modality Multi-Stream CNN for Action Recognition. *IEEE Transactions on Circuits and Systems for Video Technology*, pages 1–1, 2018.

[27] Van-Minh Khong and Thanh-Hai Tran. Improving human action recognition with two-stream 3D convolutional neural network. page 6, 2018.

[28] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Convolutional Two-Stream Network Fusion for Video Action Recognition. *arXiv:1604.06573 [cs]*, April 2016.

[29] Lin Sun, Kui Jia, Kevin Chen, Dit Yan Yeung, Bertram E. Shi, and Silvio Savarese. Lattice Long Short-Term Memory for Human Action Recognition. pages 2166–2175. IEEE, October 2017.

[30] Harshala Gammulle, Simon Denman, Sridha Sridharan, and Clinton Fookes. Two Stream LSTM: A Deep Fusion Framework for Human Action Recognition. pages 177–186. IEEE, March 2017.

[31] Moez Baccouche, Franck Mamalet, Christian Wolf, Christophe Garcia, and Atilla Baskurt. Sequential Deep Learning for Human Action Recognition. In Albert Ali Salah and Bruno Lepri, editors, *Human Behavior Understanding*, Lecture Notes in Computer Science, pages 29–39. Springer Berlin Heidelberg, 2011.

[32] Puneet Kumar, Mathias Perrollaz, Stephanie Lefevre, and Christian Laugier. Learning-based approach for online lane change intention prediction. pages 797–802. IEEE, June 2013.

[33] Ismail Dagli, Michael Brost, and Gabi Breuel. Action Recognition and Prediction for Driver Assistance Systems Using Dynamic Belief Networks. In G. Goos, J. Hartmanis, J. van Leeuwen, Jaime G. Carbonell, Jörg Siekmann, Ryszard Kowalczyk, Jörg P. Müller, Huaglory Tianfield, and Rainer Unland, editors, *Agent Technologies, Infrastructures, Tools, and Applications for E-Services*, volume 2592, pages 179–194. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[34] Julian Schlechtriemen, Andreas Wedel, Joerg Hillenbrand, Gabi Breuel, and Klaus-Dieter Kuhnert. A lane change detection approach using feature ranking with maximized predictive power. pages 108–114. IEEE, June 2014.

[35] Nachiket Deo, Akshay Rangesh, and Mohan M. Trivedi. How would surround vehicles move? A Unified Framework for Maneuver Classification and Motion Prediction. *arXiv:1801.06523 [cs]*, January 2018.

[36] Adam Houenou, Philippe Bonnifait, Veronique Cherfaoui, and Wen Yao. Vehicle trajectory prediction based on motion model and maneuver recognition. pages 4363–4369. IEEE, November 2013.

[37] David Lenz, Frederik Diehl, Michael Truong Le, and Alois Knoll. Deep neural networks for Markovian interactive scene prediction in highway scenarios. pages 685–692. IEEE, June 2017.

[38] Warren McCulloch and Walter Pitts. A Logical Caluculus of the Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biology*, 52:99–115, 1990.

[39] In The Brain and F. Rosenblatt. *The Perceptron: A Probabilistic Model for Information Storage and Organization*. 1958.

[40] Kevin Gurney. *Introduction to Neural Networks*. UCL Press, 1997. OCLC: 892785047.

[41] Rudolf Kruse, Christian Borgelt, Christian Braune, Sanaz Mostaghim, and Matthias Steinbrecher. *Computational Intelligence: A Methodological Introduction*. Texts in Computer Science. Springer-Verlag, London, 2 edition, 2016.

[42] Aditya Sharma. Understanding Activation Functions in Deep Learning — Learn OpenCV. https://www.learnopencv.com/understanding-activation-functions-in-deep-learning/ last access 09.08.2018 13:50, 2017.

[43] Ben Kröse and Patrick ver der Smagt. *An Introduction to Neural Networks*. 8 edition, 1996.

[44] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv:1609.04747 [cs]*, September 2016.

[45] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[46] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. page 30, 2014.

[47] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, April 1980.

[48] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-Scale Video Classification with Convolutional Neural Networks. pages 1725–1732. IEEE, June 2014.

[49] Blogpost. Understanding LSTM Networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/ last access 31.08.2018 11:04, 2015.

[50] Felix A Gers, Nicol N Schraudolph, and Jurgen Schmidhuber. Learning Precise Timing with LSTM Recurrent Networks. page 29, 2002.

[51] Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to Forget: Continual Prediction with LSTM. *Neural Computation*, 12(10):2451–2471, October 2000.

[52] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun Woo. Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting. *arXiv:1506.04214 [cs]*, June 2015.

[53] Simon Baker, Daniel Scharstein, J. P. Lewis, Stefan Roth, Michael J. Black, and Richard Szeliski. A Database and Evaluation Methodology for Optical Flow. *International Journal of Computer Vision*, 92(1):1–31, March 2011.

[54] Bruce Lucas and Takeo Kanade. An Interative Image Registration Technique with an Application to Stereo Vision. Vancouver, British Columbia, 1981. Carnegie-Mellon University, Computer Science Department.

[55] Berthold Horn and Brian Schunck. Determining Optical Flow, 1980.

[56] Andry Maykol G. Pinto, A. Paulo Moreira, Paulo G. Costa, and Miguel V. Correia. Revisiting Lucas-Kanade and Horn-Schunck. *Journal of Computer Engineering and Informatics*, 1(2):23–29, April 2013.

[57] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks. pages 1647–1655. IEEE, July 2017.

[58] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv:1603.04467 [cs]*, March 2016.

[59] Vadim Kantorov and Ivan Laptev. Efficient Feature Extraction, Encoding, and Classification for Action Recognition. pages 2593–2600. IEEE, June 2014.

[60] Gunnar Farnebäck. Two-Frame Motion Estimation Based on Polynomial Expansion. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Josef Bigun, and Tomas Gustavsson, editors, *Image Analysis*, volume 2749, pages 363–370. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[61] Thomas Brox, Andrés Bruhn, Nils Papenberg, and Joachim Weickert. High Accuracy Optical Flow Estimation Based on a Theory for Warping. In Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Dough Tygar, Moshe Y. Vardi, Gerhard Weikum, Tomás Pajdla, and Jiří Matas, editors, *Computer Vision - ECCV 2004*, volume 3024, pages 25–36. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[62] Deepak Pathak, Ross Girshick, Piotr Dollár, Trevor Darrell, and Bharath Hariharan. Learning Features by Watching Objects Move. *arXiv:1612.06370 [cs, stat]*, December 2016.

[63] Hengshuang Zhao, Xiaojuan Qi, Xiaoyong Shen, Jianping Shi, and Jiaya Jia. ICNet for Real-Time Semantic Segmentation on High-Resolution Images. *arXiv:1704.08545 [cs]*, April 2017.

[64] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? The KITTI vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, Providence, RI, June 2012. IEEE.

[65] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The Cityscapes Dataset for Semantic Urban Scene Understanding. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3213–3223, Las Vegas, NV, USA, June 2016. IEEE.

[66] Eder Santana and George Hotz. Learning a Driving Simulator. *arXiv:1608.01230 [cs, stat]*, August 2016.

[67] Will Maddern, Geoffrey Pascoe, Chris Linegar, and Paul Newman. 1 year, 1000 km: The Oxford RobotCar dataset. *The International Journal of Robotics Research*, 36(1):3–15, January 2017.

[68] Fisher Yu, Wenqi Xian, Yingying Chen, Fangchen Liu, Mike Liao, Vashisht Madhavan, and Trevor Darrell. BDD100K: A Diverse Driving Video Database with Scalable Annotation Tooling. *arXiv:1805.04687 [cs]*, May 2018.

[69] The Udacity open source self-driving car project. Contribute to udacity/self-driving-car development by creating an account on GitHub. Udacity, September 2018.

[70] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for Data: Ground Truth from Computer Games. *arXiv:1608.02192 [cs]*, August 2016.

[71] Iuliia Kotseruba, Amir Rasouli, and John K. Tsotsos. Joint Attention in Autonomous Driving (JAAD). *arXiv:1609.04741 [cs]*, September 2016.

[72] Gabriel J. Brostow, Jamie Shotton, Julien Fauqueur, and Roberto Cipolla. Segmentation and Recognition Using Structure from Motion Point Clouds. In David Forsyth, Philip Torr, and Andrew Zisserman, editors, *Computer Vision – ECCV 2008*, volume 5302, pages 44–57. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[73] H. Kataoka, Y. Aoki, Y. Satoh, S. Oikawa, and Y. Matsui. Fine-Grained Walking Activity Recognition via Driving Recorder Dataset. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pages 620–625, September 2015.

[74] Hirokatsu Kataoka, Yutaka Satoh, Yoshimitsu Aoki, Shoko Oikawa, and Yasuhiro Matsui. Temporal and Fine-Grained Pedestrian Action Recognition on Driving Recorder Database. *Sensors*, 18(2):627, February 2018.

[75] MOT Challenge. https://motchallenge.net/ last access 19.09.2018 15:40.

[76] Atilim Gunes Baydin, Robert Cornish, David Martinez Rubio, Mark Schmidt, and Frank Wood. Online Learning Rate Adaptation with Hypergradient Descent. *arXiv:1703.04782 [cs, stat]*, March 2017.