Interdisciplinary Center for Scientific Computing

University of Heidelberg

Analysis of a traffic-scenario based on autonomous driving datasets

Bachelor Thesis in Physics
submitted by

Paul Elias Gondolf

born in Pforzheim (Germany)

2020

This Bachelor Thesis has been carried out by Paul Elias Gondolf at the
Interdisciplinary Center for Scientific Computing in Heidelberg
under the supervision of
Prof. Dr. Carsten Rother.

Abstract

Critical traffic situations are a challenging part of the autonomous driving task, not only due to the legal and ethical questions they implicate but also because they are scarce and often quite complex and hence arduous to study. However, with the availability of autonomous driving datasets, providing an increasing amount of sensor data, and recent advances in machine learning, allowing for easier meta-data extraction, the analysis of such scenarios is now more promising than ever.
Thus in this work, we present the study of the specific scenario of a pedestrian crosswalk managed by a traffic light system, where a critical situation appears when pedestrians are on the street during the green phase of the vehicle traffic light. Deploying deep-learning models on DriveU and Cityscapes and using the ground truth of the two datasets we estimate the corrected probability of an alike setup with 0.0825 % compared to setups with different light phases and no pedestrians on the street. We also provide the true-positive real-world samples of the critical scenarios we found in the datasets and discuss possible flaws of our analysis, with its approximated 17.87 % false-positive error rate.

Zusammenfassung

Kritische Situationen im Straßenverkehr sind eine der Herausforderung die nicht nur wegen der etischen und rechtlichen Fragen die sie implizieren eine hohe Relevanz für das autonome Fahren besitzen, sondern auch weil sie aufgrund ihrer Seltenheit und Komplexität schwierig zu analysieren sind. Mit der Verfügbarkeit von größeren Datensätzen, die ein immer umfangreicheres Maß an Sensordaten zur Verfügung stellen und Fortschritten im Bereich des maschinellen Lernens, sind die Voraussetzung für das Studium solcher Situationen heute allerdings sehr vielversprechend.
Wir stellen deshalb in dieser Arbeit das Scenario eines ampelgeregelten Fußgängerüberwegs vor, wobei eine kritische Situation entsteht, wenn sich ein Fußgänger während der grünen Ampelphase auf der Straße befindet. Wir verwenden Neuronale Netze auf DriveU und Cityscapes und die "Ground Truth" der beiden Datensätze und schätzen damit die Wahrscheinlichkeit des beschriebenen Szenarios auf 0.0825 % im Vergleich zu Szenarien mit andere Lichtphasen und keinen Fußgängern auf dem Überweg. Wir geben außerdem die gefunden wahr-positiven Beispiele eines kritischen Szenarios aus den Datensätzen an und diskutieren mögliche Schwächen unseres Ansatzes mit seiner 17.87 prozentigen falsch-positiven Rate.

Contents

1 Introduction

1.1 Motivation

There is hardly anything more important for the development of the world we know today than the automation in all major areas of our private, social, and economic life. In the form of algorithms, automation operates aeroplanes, production lines, whole industrial facilities, and maybe even the domestic coffee maker. While advancing over the years, we gave those algorithms more and more responsibilities and used them in new and more complex environments. Nowadays their tasks are not limited to serve milk warm or cold, or coffee with or without caffeine, but they already actualize futuristic visions from the science fiction novels of the last century. One of those innovative ideas that are on the verge of realisation is autonomous driving. This means cars and trucks that navigate traffic without any human intervention or even supervision while succeeding their counterparts out of flesh and blood in efficiency and safety. Accomplishing this task the algorithms first have to evaluate the vast amount of their sensor data and extract the relevant information and second use the gained information to plan their next move. With the state space as well as the action space being high dimensional and hard to grasp the research in this field tends towards approaches with dynamic programming and machine learning to tackle complexity. Algorithms of this kind heavily rely on extensive, diverse as well as representative datasets to prepare them for every eventuality and especially critical situations. However, this latter situations are qua definition very sparse and hence also rarities in the training data. The frequency of their appearance hereby stands in flagrant contrast to their importance for supervised learning and autonomous driving in general, since it is only juridically and ethically justifiable to let go a driverless vehicle if it is capable of acting in akin situations. Thus one first step towards more applicable algorithms is the study of critical scenarios their characterisation and statistical evaluation, answering the questions: How does a critical scenario look like in reality and how often do they appear?

The prior considerations formed the motivation for this thesis, whereas this work wants to take a closer look at a specific conflicting situation. Conflicting means, that the environment actively (by a signal, i.e. a traffic light or sign) indicates a certain predefined configuration of itself, while the reality hardly corresponds. This active indication stands out, especially in the context of autonomous driving, since a traffic participant, for example, the autonomous driving system, has to realise the discrepancy between reality and the indicated state, although he might be used to accurate signals. Now the approach of this thesis is to search urban- and suburban-scene datasets for our chosen scenario, and finally give estimates for the probabilities regarding the appearance of critical in contrast to uncritical setups. Therefore our contribution is showing that such an analysis is possible and present an approach on how to conduct it. In addition to that, the estimates, as well as the real-world examples we provide, could be used to build synthetic instances for training models as well as studying the behaviour of autonomous driving algorithms to the chosen conflict.

1.2 Specification and Goals

The scenario which was picked for this thesis is rather common in urban and suburban domains and therefore allows a more accurate analysis due to a rather high representation in already existing traffic datasets. Because of its rigid structure, the scenario is also well detectable.
The scenario is one of a pedestrian crossing that is managed by a traffic light system. The actors in this scenario are the pedestrians and the vehicle, whereat all information about the environment is processed from the vehicles point of view.
Now in a normal setup, the traffic light manages the pedestrian crossing and prevents the collision of humans and vehicles by giving visual signals to both parties. A conflict, however, appears if pedestrians behave transgressively and are on the street while the traffic light signals the vehicle a free crosswalk.
As we stated in the motivation, the goal of this thesis is to decompose this scenario and condense the information from the datasets, finally giving estimates for the probability of the possible setups determined by the state of the managing traffic lights and the presence of pedestrians on the crossing.
Here the challenge of this analysis lies in the parsing of the urban- and suburban-scene dataset, since the meta-information that is relevant for the calculations has to be extracted from real-world and disparity images and the ground truth of the data. For every sample, there has to be an examination if it contains a pedestrian crossing, if people are walking over it and which state the managing traffic light is.

1.3 Structure of the Thesis

In the beginning, we discuss foundations of machine learning in general and then take a closer look at supervised learning and in particular convolutional neural networks that play a big role when it comes to analysing image and video data. This is followed by a discussion of related work concerning the topic of traffic light classification and detection as a paragraph about the rise of urban- and suburban-scene datasets, being essential to the proposed analysis. After a short introduction of the used datasets we come to the four main parts of the thesis:

In the first part a convolutional neural network named DenseNet is trained and tested on traffic light images from the DriveUC dataset to classify state and type of the respective samples. Furthermore, a model for the identification of relevant traffic lights is developed and then trained and tested on DriveUC as well. The latter model uses the state, type, measures and position of a traffic light to decide about its relevance. Finally both models are merged to form a traffic light classification pipeline, which is then evaluated on the DriveUC test set.

In the second part we shortly introduce the Panoptic-Deeplab model, being a panoptic-segmentation pipeline pre-trained on the Cityscapes train dataset. We do add some postprocessing operations to this pipeline and discuss its results on the Cityscapes extra, test and the DriveU dataset.

In the third part we build a set of rules formalising the identification of pedestrian crossings and detecting the presence of pedestrians on this crossings in images of urban- and suburban-scenes. Those rules are based on the depth information, a panoptic-segmentation of the image as well as type, state and relevance labels for all traffic lights present. For both datasets, we do not have the full input and hence have to extract this information from the data accessible to us. The Cityscapes train dataset provides a ground truth of panoptic-segmentation and the disparity, however, is missing the labels of the traffic lights. Here the model of part one comes into play providing us with type, state and relevance labels for all traffic lights. For DriveU we can, on the other hand, rely on labelled traffic lights and depth information but need to generate the panoptic-segmentation using the pre-trained Panoptic-Deeplab pipeline of part two. Finally with Cityscapes train-extra and test we have neither the segmentation nor the type and state information, hence we utilise both, our Panoptic-DeepLab as well as the traffic light classification model.

In the fourth part the condensed information is used to calculate the probabilistic estimates and we analyse some outputs of the pipeline to get aware of the flaws of our approach.

In the end, we conclude our results and discuss the ability of the used approaches. We reason about the accuracy of the calculated estimates and then finish with some thoughts about future work in this field.

2 Foundations

2.1 Machine Learning

A definition of a machine learning problem is given by Thomas Mitchell in [1]:

"Each machine learning problem can be precisely defined as the problem of improving some measure of performance P when executing some task T, through some type of training experience E."

The generality of this definition leaves a lot of room for details and characterisations of the abstract entities P, T and E and we, therefore, limit ourselves to the common machine learning problems which capture the most attention in current research and technologies.
In this problems P, T and E are all incorporated in an algorithm, in which P is given as some quantitative performance measure tailored to the task at hand. The tasks controlled by P is then a rule specifying the processing of a sample, which is normally represented by a vector $\boldsymbol{x} \in \mathbb{R}^n$. Now a sample itself can be divided up into its features being objects, measurements or events that are in some way interconnected with the task, where the i^{th} feature corresponds to the i^{th} entry of $\boldsymbol{x}$. If we have a set of such samples we usually talk about a dataset.
By the experience that the algorithms gain we can broadly divide them into two groups, with however a quite fluent border between them:

The unsupervised learning. In the unsupervised learning setting, we are given a dataset and want to gain experience in the form of knowledge about the underlying structure of the data by learning its probability distribution or just properties of this distribution.

The supervised learning. In the supervised learning setting we are given a dataset as well but in this case, a sample, besides the regular features, also contains several special features that we call labels or targets. It is common to separate the regular features and the labels/targets and represent the first one with $\boldsymbol{x}$ and the latter one with $\boldsymbol{y}$. With this definition a sample now consists of the tuple $(\boldsymbol{x}, \boldsymbol{y})$, and from now on we call $\boldsymbol{x}$ only the features and $\boldsymbol{y}$ the targets/labels. With these definitions, the goal of supervised learning is to gain experience in the form of some mapping from $\boldsymbol{x}$ to $\boldsymbol{y}$.

A more sophisticated overview of machine learning is given in [2, pp. 96-105], which also was used as a basis for this part.

2.2 Supervised Learning

In this thesis we are only[1] confronted with supervised learning problems and we, therefore, want to discuss some more details and establish basic terminology. It is quite convenient and more vivid to develop

[1]We neglect the application of DBSCAN in sec. 6.

this terminology while dealing with a concrete model. As we are utilising a feed-forward neural network (FNN) later when classifying the relevance of traffic lights it seems reasonable to choose this model and the classification task with i.i.d. data as representatives.

2.2.1 Classification

We start with the dataset give as:

$$\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i) \mid i \in [\![1, n]\!]\} \subseteq \Omega_X \times \Omega_Y \subseteq \mathbb{R}^n \times \mathbb{R}^m \tag{2.1}$$

together with the assumption that there exist dependencies between $\boldsymbol{x}$ and $\boldsymbol{y}$ that allow for the modelling of a function:

$$f : \Omega_X \to \Omega_Y, \quad \boldsymbol{x} \mapsto f(\boldsymbol{x}) = \hat{\boldsymbol{y}} \tag{2.2}$$

Now a classification problem has the property that our space of labels is finite $|\Omega_Y| < \infty$, while the space of features can be of arbitrary cardinality. One should also notice the hat in eq. 2.2 clarifying that this $\hat{\boldsymbol{y}}$ is the prediction of our model what $\boldsymbol{y}$ might be and not the actual $\boldsymbol{y}$. This is a quite important fact as it implicitly indicates our belief that our data is i.i.d. and drawn from a probability distribution, thus, in general, we can not learn a deterministic function but only an estimator for $\boldsymbol{y}$.
Since we believe both $\boldsymbol{x}$ and $\boldsymbol{y}$ are realisations of random variables, Ω_X and Ω_Y can be interpreted as probability spaces and there should exist a joint distribution:

$$p : \Omega_X \times \Omega_Y \to [0, \infty], \qquad (\boldsymbol{x}, \boldsymbol{y}) \mapsto p(X = \boldsymbol{x}, Y = \boldsymbol{y}) \tag{2.3}$$

Where with the capital letters we indicate random variables and with the small letters realisations. Now with the bias rule and the factorisation of joint probabilities [3, pp. 14, 22] we obtain:

$$p(Y|X) = \frac{p(X, Y)}{p(X)} \tag{2.4}$$

If we were in the possession of $p(Y|X = \boldsymbol{x})$ the most intuitive estimator would be:

$$f(\boldsymbol{x}) = \arg\max_{\boldsymbol{z} \in \Omega_Y} \; p(Y = \boldsymbol{z} | X = \boldsymbol{x}) \tag{2.5}$$

which means we predict the $\boldsymbol{z}$ that maximises the posterior probability given our input $\boldsymbol{x}$. This definition of f is called the bayes classifier and is the optimal classifier when trying to minimise the risk on the 0-1-loss [3, pp. 39–42]:

$$R(f) = \mathbb{E}_X \mathbb{E}_{Y|X} L(Y = \boldsymbol{y}, f(X = \boldsymbol{x})) \tag{2.6}$$

with

$$L\left(Y = \boldsymbol{y}, f(X = \boldsymbol{x})\right) = \begin{cases} 0 & f(X = \boldsymbol{x}) = \boldsymbol{y} \\ 1 & f(X = \boldsymbol{x}) \neq \boldsymbol{y} \end{cases} \tag{2.7}$$

being subject to

$$\min_f R(f) \tag{2.8}$$

Here $\mathbb{E}[\cdot]$ denotes the expectation subject to its argument and the regarding probability variable written in the lower index position.

Now the goal of classification (and also regression) is to either learn f directly or learn some approximation of $p(Y|X)$ and then set for example eq. 2.5 as an estimation function. The learning of the latter can happen by directly modelling $p(Y|X)$, which is called the discriminative approach, or by modelling the joint distribution (eq. 2.4) or likelihood and marginals going by the name generative approach.

2.2.2 The Feedforward Neural Network

If we were in possession of $p(Y|X = \boldsymbol{x})$ our problem would be solved as we could just use eq. 2.5 for our estimator f. As we do not know the underlying probability distribution, we have to think about a way obtaining a good enough approximation of $p(Y|X = \boldsymbol{x})$ and here our feed-forward neural network (FNN) comes in to play. We choose the discriminative approach and just model the posterior with such a FNN:

$$\tilde{p}_{\boldsymbol{\theta}}(Y|X = \boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\theta}) \tag{2.9}$$

where $\boldsymbol{\theta} \in \Theta \subset \mathbb{R}^l$ is the parameter vector of our FNN in the parameter space. One notices that the network not only outputs the probability for a certain $\boldsymbol{y}$ but rather the whole distribution given $X = \boldsymbol{x}$. With the set Ω_Y being finite we can enumerate it and eq. 2.9 not only can be understood as probability table but can be brought into a vector shape, with the i^{th} entry corresponding to the probability of $\tilde{p}_{\boldsymbol{\theta}}(Y = \boldsymbol{y}^{(j)}|X = \boldsymbol{x})$. Here $\boldsymbol{y}^{(j)}$ is the j^{th} element of the enumeration of Ω_Y.

Our FNN is now on the one hand dependent on a vector of parameters representing the "learnable part" and on the other hand on its architecture, meaning the number and sizes of its layers. If we call $\boldsymbol{h}_{i-1} \in \mathbb{R}^{n_{i-1}}$ the input to the i^{th} layer and $\boldsymbol{h}_i \in \mathbb{R}^{n_i}$ the output with $i \in \mathbb{N}$ then we can describe this layer as a affine transformation followed by a non linear activation function $g_i : \mathbb{R}^{n_i} \to \mathbb{R}^{n_i}$:

$$\boldsymbol{h}_i = g_i\left(\boldsymbol{W}_i(\boldsymbol{h}_{i-1})\right) \tag{2.10}$$

Here we made use of the fact that every linear transformation over finite $\mathbb{R}$-vector-spaces can be represented by a matrix ($\boldsymbol{W}_i \in \mathbb{R}^{n_{i-1} \times n_i}$) if the bases are specified. Furthermore we also incorporated the shift of the origin into the matrix-vector product by implicitly adding another entry at constant one to the vector $\boldsymbol{h}_{i-1}$. The so defined matrices are the constituents of our parameter vector $\boldsymbol{\theta}$. We depicted a schematic drawing of a layer in fig. 1 for a better understanding.

With this definition a FNN is now nothing more than a concatenation of an input and an output layer and some arbitrary number of layers of arbitrary shape in between, often referred to as hidden layers. Hence to complete our construction of an FNN, we just have to choose activation functions and the number and sizes of the hidden layers we want to use (the input and output dimensions are fixed by our feature vector $\boldsymbol{x}$ and the cardinality of Ω_Y respectively).

We do not want to talk too much about activation functions but just introduce the heavily used function that has proven to be the most effective in most cases [4], the rectifying linear unit (ReLU):

$$ReLU : \mathbb{R} \to \mathbb{R}, \qquad x \mapsto ReLU(x) = \max\{x, 0\} \tag{2.11}$$

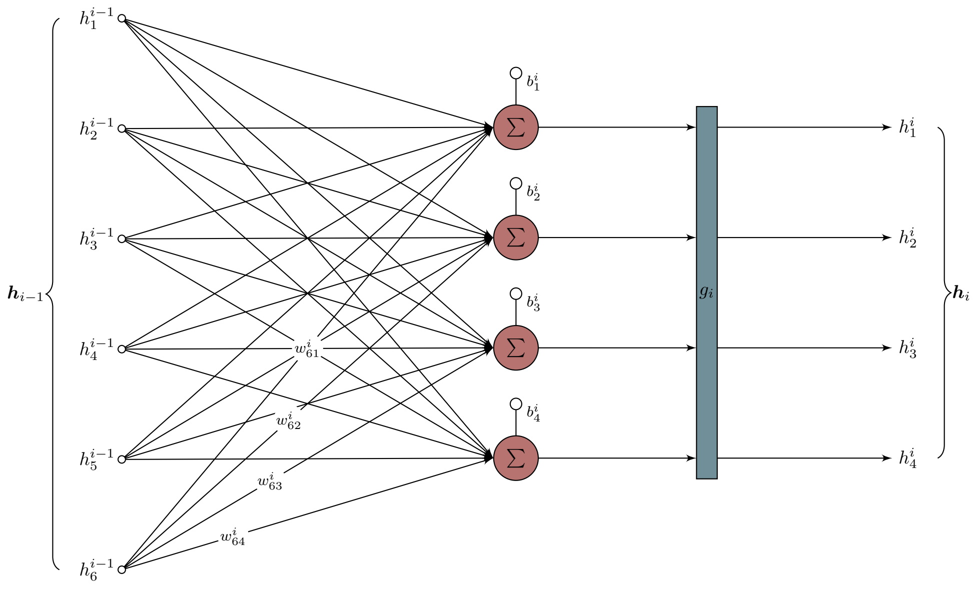


Fig. 1: A scheme of the i^{th} layer with a vector of dimension six as input and dimension four as output. The features h_k^{i-1} of the input vector are multiplied with the weights of the layer w_{kl}^i and then summed up together with a respective explicit bias term b_l^i. For the sixth input, the weights are written on the arrows to visualise the principle. The affine transformation is then followed by the activation function of the layer g_i, which finally produces the output.

It uses no parameters and is applied pointwise to the vector entries. Its benefits are mainly its simple derivative (setting it to zero at the origin) facilitating computation and especially back-propagation as well as its linearity, precluding divergence of features in deeper layers.
This function, however, can not be used as an activation function in the last layer if we want to keep our probability interpretation of the output. For the last layer, we, therefore, apply the so-called normalised exponential function in short softmax that is defined as:

$$softmax : \mathbb{R}^n \to [0,1]^n, \qquad \boldsymbol{z} \mapsto softmax(\boldsymbol{z}) = \frac{e^{\boldsymbol{z}}}{\sum_{j=1}^n e^{z_j}}$$

Whereby the exponential function is applied pointwise to the vector in the argument and z_j is the j^{th} entry of $\boldsymbol{z}$. The vector produced by the softmax has the property that all its entries are non-negative and sum to one, implying we can interpret them as probabilities.
The specifics about layer dimensions and the depth of the network is now heavily dependent on the specific task and therefore not of greater interest to us. We rather want to discuss another centrepiece of machine learning and that is how to accumulate experience or in other words how to learn our approximation of $p(Y|X)$ by adjusting our parameters $\boldsymbol{\theta}$.

2.2.3 Learning and the Loss

The loss and the learning go hand in hand as the loss is our measure P that quantifies the performance of our model and therefore the learning process. Sampling $\boldsymbol{y}$ given $X = \boldsymbol{x}$ follows a categorical distribution due to the finite space of classes [3, pp. 444–448]. This however means that the likelihood of our dataset is:

$$l(\mathcal{D}) = \prod_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} p(Y = \boldsymbol{y}|X = \boldsymbol{x}) \tag{2.12}$$

with the factorisation only being possible since we assumed the data is i.i.d. Now the problem is still that we are not in possession of the true distribution but only of the dataset, thus instead of using the true distribution in the likelihood we use our model. We then argue that the data should maximise this likelihood and choose $\boldsymbol{\theta}$ accordingly:

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}\in\Theta} \prod_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \tilde{p}_{\boldsymbol{\theta}}(Y = \boldsymbol{y}|X = \boldsymbol{x}) \tag{2.13}$$

The $\hat{\boldsymbol{\theta}}$ is called the maximum likelihood estimate (MLE) of our $\boldsymbol{\theta}$ and is in a more general manor discussed in [2, pp. 129–130]. We can utilise the logarithm to transform eq. 2.13 into a sum, while both optimisations are equivalent due to the concavity of the logarithm on $(0, \infty)$. Furthermore we added a minus to make it a minimisation problem:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}\in\Theta} - \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \log\left(\tilde{p}_{\boldsymbol{\theta}}(Y = \boldsymbol{y}|X = \boldsymbol{x})\right). \tag{2.14}$$

The argument of argmin is also called the negative log likelihood (NLL) [5, p. 218]. With our estimator now defined, we are confronted with two problems. The first one is our finite and maybe unrepresentative dataset and the second one is the optimisation of our model, meaning getting it close to the true distribution which often is not equivalent to simply optimising eq. 2.14. Both problems are quite complex, task specific and still part of active research, with for example the approach to generate new data points [6] to better understand data generating processes and inflate existing datasets. For this reason we omit the issues we might run into with our dataset completely and just briefly present an optimisation algorithm called Adam [7] that we later want to use while referring to [2, Chapter 7-8] for a deeper insight.

Adam is a stochastic gradient descent algorithm with adaptive estimates of lower order moments. It is based on the so called gradient descent algorithm which is an optimisation algorithm that uses the gradient of the objective function, in our case:

$$\mathcal{L}_\lambda(\boldsymbol{\theta}) = -\frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \log\left(\tilde{p}_{\boldsymbol{\theta}}(Y = \boldsymbol{y}|X = \boldsymbol{x})\right) + \lambda\|\boldsymbol{\theta}\|_2, \tag{2.15}$$

to find a global minimum by making parameter alterations in the opposite direction of this gradient (i.e. in the direction of the steepest decent) until convergence. One notices that we multiplied eq. 2.14 with a factor having the effect of averaging the gradients over all samples. We also added a term by which we want to control the magnitude of our parameters. This practice is called L_2 regularisation and is one of numerous tactics to account for overfitting, meaning that our model over adapts to the given samples of our distribution (i.e. our dataset) rather than to the distribution itself. Overadaptation is often unwanted since in many cases it results in bad performance on previously unseen data. The parameter $\lambda \in \mathbb{R}_0^+$ in this context is called weight decay parameter.

It has proven to be effective not to calculate the gradient for the whole dataset but instead uniformly and without replacement sample fixed size subsets, so called mini batches from the training set. This subsets

are then iterated threw, calculating the gradient of the objective function for the respective subset and making according alterations to the parameters in every step [2, pp. 151–153]. This practice goes by the name stochastic gradient descent while the size of the mini batches is called batch size and one iteration threw all mini batches (i.e. the whole dataset) is named epoch.
The Adam algorithm (alg. 3) builds on this while not only utilising the gradient of the current time step but also calculating individual first and second moment estimates from previous timesteps making the stochastic gradient descent algorithm more efficient. For the mathematical properties of the algorithm see [7].

2.2.4 Convolutional Neural Networks

Convolutional neural networks (CNNs) are just another type of model, working with different layer types compared to the FNNs described above. However, the goal of approximating $p(Y|X)$ remains and the foundations of classification and learning, therefore, still are valid. Hence it is sufficient to just to discuss the basic building blocks of convolutional neural networks in this section, while optimising the parameters is done the same as with a FNN.

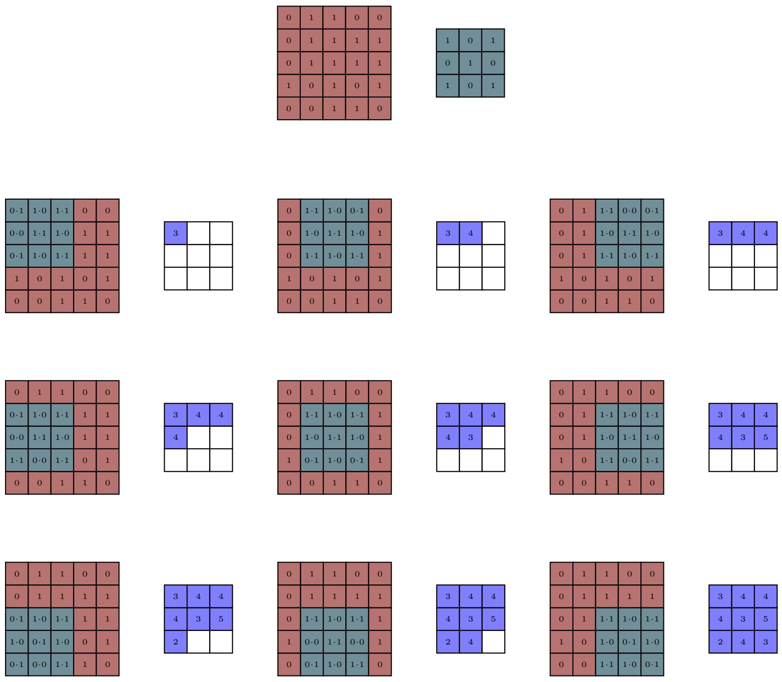


Fig. 2: A two-dimensional convolution performed on a two-dimensional input. The graphic shows every consecutive operation with the kernel moving over the input constructing one output by summing up all the values of interest in every step. In this example, we do not pad the input (meaning adding values to the sides of it) and use a stride of $(1, 1)$ resulting in one step along the first tensor axis and one along the second with the kernel moving along the tensor axes in their given order.

The first building block would be a simple convolutional layer, which is quite similar to eq. 2.10 with the only difference that $\boldsymbol{K_i}$ is now a tensor of arbitrary shape called Kernel, and we also no longer perform a matrix multiplication but a convolution:

$$\boldsymbol{H}_i = g_i\left(\boldsymbol{K}_i * \boldsymbol{H}_{i-1}\right). \tag{2.16}$$

Our activation function remains and is still in most cases a ReLU while the parameter vector of our model is formed by this kernels $\boldsymbol{K}_i$. Now a convolution allows for differently shaped input as to the FNNs which, due to the matrix-vector product, could only process vectors. The convolution operation, however, is defined for arbitrarily shaped tensors ($\boldsymbol{H}_i$), with the restriction that the number of dimensions of kernel and input must match. This makes CNNs interesting for computer vision and especially image classification, where a sample (e.g. an image) is naturally represented by a tensor of shape $C \times W \times H$, namely the number of channels a width and height. In this context it is common to talk about a tensor having C channels of $W \times H$ feature maps. The convolution operation is not only chosen out of convenience but also for incorporating a concept that we know is intrinsic to some data (e.g. image data) and that is the strong covariance of neighbouring[2] entries in a sample. A simple example of a convolution is depicted in fig. 2.

Another building block is the pooling operation. That is a parameter-free downsampling operation which is quite similar to a convolution (mean pooling is indeed a convolution). It takes a tensor as input and starts with a specified entry calculating the maximum, sum or mean overall entries that lie in a defined distance (sometimes referred to as window) to generate the first entry of the output. In this context, distance is commonly defined for every tensor axis individually. Done with the first entry this filter moves to the next, following the given stride, repeating the above operation. The stride, for pooling as well as for the convolution, defines a rule on how to parse the image. It is normally given as "step size" along each tensor axis, with the tensor axis being parsed in their given order. Therefore, this stride, together with the padding and distance (or for convolutions the kernel size), defines the shape of the output in the end. The mathematical concepts of pooling, convolutions, stride and padding are more thoroughly discussed in [2, Chapter 9].

The last operation we briefly want to introduce is the batch normalisation, proposed in [8]. If we consider an input to our batch normalisation layer of dimension d say $\boldsymbol{h} = (h_1, \dots, h_d)$ and a mini batch $\mathcal{B} = \{\boldsymbol{h}^{(1)}, \dots, \boldsymbol{h}^{(m)}\}$ of such inputs, then the layer performs the following operations:

$$\boldsymbol{\mu}_{\mathcal{B}} = \frac{1}{m}\sum_{r=1}^{m} \boldsymbol{h}^{(r)} \tag{2.17}$$

$$\boldsymbol{\sigma}^2_{\mathcal{B}} = \frac{1}{m}\sum_{r=1}^{m} \left(\boldsymbol{h}^{(r)} - \boldsymbol{\mu}_{\mathcal{B}}\right)^2 \tag{2.18}$$

$$\hat{\boldsymbol{h}}^{(r)} = \frac{\boldsymbol{h}^{(r)} - \boldsymbol{\mu}_{\mathcal{B}}}{\sqrt{\boldsymbol{\sigma}^2_{\mathcal{B}} + \boldsymbol{\epsilon}}} \tag{2.19}$$

$$\boldsymbol{y}^{(r)} = \boldsymbol{\gamma} \odot \hat{\boldsymbol{h}}^{(r)} + \boldsymbol{\beta} \tag{2.20}$$

Here $\boldsymbol{y}$ is the output of the layer, $\boldsymbol{\gamma}$, $\boldsymbol{\beta}$ are learnable parameter vectors of dimension d (meaning they are also part of $\boldsymbol{\theta}$) while $\boldsymbol{\epsilon}$ is a vector filled with small values added for numeric stability. $\odot$ denotes the

[2]Neighbouring is an abstract term here and its meaning can differ depending on the data. In the context of images, the term indeed refers to neighbouring pixels.

pointwise product while the power operation in eq. 2.18 is as well pointwise.

By normalising the layer inputs inside the network and introducing a learnable mean and standard deviation the batch normalisation allows for accelerated computations as well as often improving the performance of a neural network. The reasons for this positive impact are actively discussed in the deep learning community with several opinions about where it stems from. The original paper [8] assumed that batch normalisation would mitigate the change of the layers input distributions during training called internal covariate shift while some argue that it has a smoothing effect on the loss surface [9] and others believe it decouples the length and direction of the parameters [10]. On one thing they, however, all agree and this is the empirically proven beneficial effect of batch normalisation during training and testing.

2.2.5 The Implementation

The actual implementation of the above concepts is a topic of itself and we do not want to discuss it here. However, we do want to emphasise, that with the rise of more advance, and high-level toolkits [11]–[13] the implementation of models, losses and especially back-propagation has become increasingly easy. We also want to give credit to the toolkit we are heavily using to conduct our experiments, going by the name PyTorch [13]. The DenseNet model [14], as well as a lot of the used loss and optimisation algorithms already come with this library. The PyTorch toolkit and all our experiments are written in Python 3.6.9. [15].

3 Related Work

Our approach of investigating a specific traffic scenario in urban and suburban domains for the goal of better understanding the statistics and real-world realisations of critical situations in traffic has, to the best of our knowledge, not been done before. A reason for that might be that although the common configurations are frequent in reality and therefore also in urban- and suburban-scene datasets, the critical ones are not. Meaning that the analysis would indeed be possible, but hardly lead to interesting results, if the data available is too scarce, the extraction of the metadata claims too many resources or is not even possible due to missing information and quality.With the ever-increasing number of public datasets and the growing amount of meta-information (besides the raw image data) coming attached, the named problems are mitigated if not even eliminated, making the proposed analysis quite promising.

3.1 Urban- and Suburban-Scene Datasets

This rise of available urban and suburban scene datasets started with the CamVid and the LaRA dataset. The first dates back to 2008 and provides 700 instance-segmented images all recorded out of a vehicle in the city of Cambridge in England [16]. The latter was published by a French research group in 2010 recorded in the city of Paris. It contains 9.168 annotations in the form of bounding boxes and four labels assigned to the vehicle traffic lights [17]. Three years later in 2013 KIT (Karlsruhe Institute of Technology) released the KITTY dataset consisting of six hours traffic scenarios recorded with a variety of sensor modalities (e.g. colour and grayscale cameras, a GPS receiver and distance measurements provided by a laser scanner) in addition to object annotations of traffic participants in the form of 3D tracklets [18]. Another dataset with the name Daimler Urban Scene was published in the same year comprising of 5.000 stereo images of which 500 come with pixel-level class annotation from a total of five classes [19]. Then in 2015, the University of California released the so-called LISA dataset which comes with 51.826 traffic light annotations and seven labels, recorded in the city of Sun Diego [20], [21]. Following one year later in 2016 was the Cityscapes dataset containing 25.000 images together with GPS and disparity data providing hand labelled pixel- and instance-level segmentation on roughly 3.500 of this images and further coarse annotations on 20.000 of them (for the test set neither fine nor coarse annotations are publicly accessible) [22]. In the following year, a collaboration of Bosch and the Czech Technical University in Prague published the Bosch-Small-Traffic-Light-Dataset (BSTLD), comprising of 5.000 images for training and a video-sequence of 8.334 frames for evaluation, both recorded in the United States. The dataset has a ground truth of 24.242 hand-labelled traffic light annotations, in the form of bounding boxes, together with an assigned label from a total of fifteen classes in the training set and four in the evaluation set [23]. At last, in 2018 the University of Ulm released the DriveU Traffic Light Dataset (DTLD), which is up to this day the biggest publicly available traffic light dataset with 232.039 annotations from 344 unique classes. The dataset was recorded in eleven German cities and comes with

calibration and disparity data, as well as geolocation and vehicle velocity information [24].
More datasets contain urban- and suburban-scenes, for instance, COCO [25] or the dataset described in [26], however, they are either in private hands, as it is the case for the latter, or address a more general setting as it is the case for the first.
An analysis of all named dataset would be to time and resource-consuming, due to the process of extracting necessary meta-information from the raw datasets, hence we chose the two leaders in terms of samples, namely Cityscapes and DriveU for our purposes. The loss in data we face, when discarding all the other datasets, is not that severe, as both, DriveU and Cityscapes, outnumber their predecessors at least by a factor of ten. They also have some more, but also very significant advantages we will take a closer look at in sec. 4.

3.2 Traffic Light Identification and Classification

Albeit the analysis we are planning has, to our knowledge, not been done before there exists scientific research related to the methods we intend on using, to extract information from the datasets. Besides the panoptic segmentation and depth of the scenes, we are reliant on the information about the state, type and relevance of the traffic lights, hence a major part of this thesis will be devoted to the development of a traffic light classifier.
Those traffic light classifiers face a great interest of scientists and companies, as they are a substantial part of every autonomous or part-autonomous driving system. In this context, however, they are often included in a traffic light identification and classification pipeline that localises as well as classifies them. For our purposes, however, classification is sufficient, as the position is contained in the ground truth of the used datasets. Our model has a different main focus, namely the classification, and in particular the number of classes we want to classify, including type, state and relevance information. In contrast to that early approaches solely focused on the state of the traffic light (e.g. green) [27], and only more recent works classified the type (e.g. right arrow) as well, while the relevance feature, we intend to include, is merely accounted for in the newest publications [28].
The approaches taken in the field of traffic light classification, also called recognition[3], are very diverse and range from methods of classifying the winner of a majority pixel count [29], using the nearest neighbour method on Gabor image features [30], to support vector machines, applied to transformed representations of the traffic light images [31]–[33]. Some earlier works also made strong assumptions about the shape and/or colour of the traffic lights and their surroundings and used adaptive template matching to classify the state and in some cases the type too [27], [34], [35]. Newer approaches, due to the accessibility of vast amounts of data (sec. 3.1) and the rapid increase of performance of neural networks [36]–[38], mainly rely on deep learning, to classify, but also identify, the traffic lights in urban- and suburban-scenes. A typical pipeline setup for this methods would be the localisation of the traffic lights in the image by an interest-point detector, followed by classification on these regional proposals utilising a shallow [39] or a convolutional neural network [23], [40], [41]. Yet, there are also works, that perform the identification and

[3]We will use both terms interchangeably.

classification procedure with one network [26], making a careful design of pre-segmentation algorithms obsolete.

Following the overall trend towards deep learning, we also use neural networks to conduct our classification. However, with our priorities differing from the ones in the papers described, since deployment on a vehicles GPU/CPU is not part of our goal, we harness a bigger network, frequently used in image classification task like ImageNet [36], where size and speed are lesser of a limiting factor. This network by the name DenseNet [14], is indeed only a part of our pipeline and classifies solely type and state of the traffic lights. The relevance feature is predicted by a separate simple feed-forward neural network, being fed the type and state as well as locality and geometry information. The research concerning the recognition of the relevance feature is currently still quite scarce and there is, to the best of our knowledge, no approach doing a robust relevance classification. Some models use the relevance feature, however, not classifying it but using precomputed prior maps that already include the relevant traffic lights [28], [42].

4 The Datasets

In this section, we give a short introduction to the used datasets and reason about selecting them for this thesis. We also describe and introduce the modified DriveU dataset called DriveUC, used to train our traffic light classifier.

4.1 DriveU

DriveU, published in 2018, is one of the biggest open access datasets providing human labelled traffic lights in suburban and urban domains on a scale of 230.000 annotations. The images were taken in eleven German cities during the daytime. In addition to the annotations, disparity data is provided, which allows for pixel depth estimation. The label of one sample includes information about relevancy of the traffic light, its installation orientation (e.g. horizontal), the number of the light units, its state (e.g. green) and the type (e.g. left arrow) as well as some other properties. However, for our purposes the state, the relevancy and the type classes are sufficient. We decided for DriveU because it is one of the

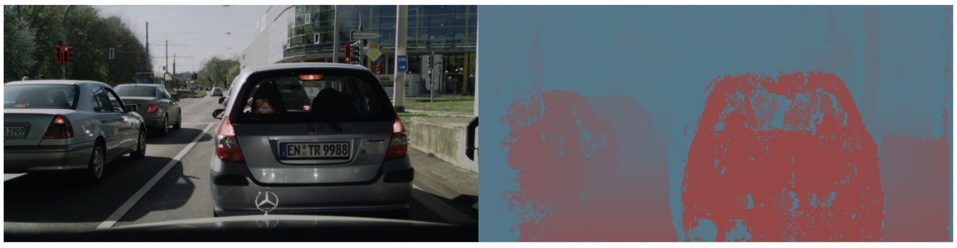


Fig. 3: A sample from the DriveU dataset. On the right-hand side is the real world image and on the left-hand side the disparity image. The bounding boxes of the traffic lights are annotated with red colour in the real-world image.

biggest open-source datasets on traffic light classification with a clear and consistent labelling policy. It also provides meta information for every image, such as disparity data, geolocation and vehicle velocity. The last but very important benefit of DriveU is its similarity to Cityscapes, a property that is discussed in sec. 4.2. In fig. 3 we depicted a sample from DriveU together with its disparity map. For a more comprehensive introduction to the dataset, we refer to its paper [24].

4.2 Cityscapes

The Cityscapes dataset is a large scale urban scene dataset for pixel-level and instance-level semantic segmentation from the year 2016. It was recorded in the streets of fifty different cities in Germany, during daytime and contains 5.000 high-level annotated images and further 20.000 coarse annotated images. In addition to the instance segmentation corresponding disparity data is existing to calculate pixel depth.

The labels consist of 30 classes of which we only need the person label, the vehicle and some environment labels (road, sidewalk, ground, terrain, parking). We choose the dataset because it is the only

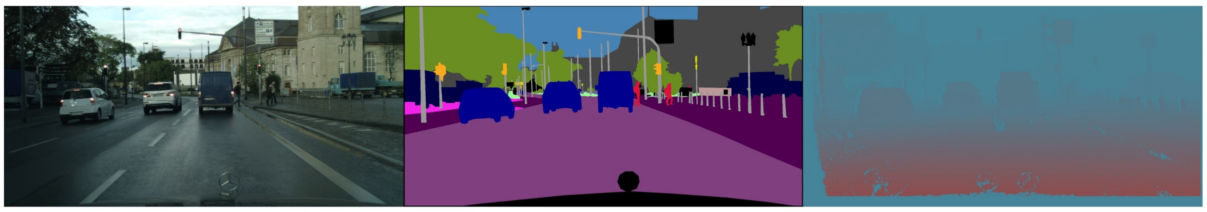


Fig. 4: A sample from the Cityscapes train dataset. On the left-hand side is the real world image, in the middle the instance segmentation and on the right-hand side the disparity.

panoptic-segmented dataset of urban scenes with disparity information on such a high number of images. Furthermore, the high-quality segmentation of the dataset is essential for this project, since we heavily rely on this segmentation when localising pedestrians and traffic lights. On the one hand, we can use the ground truth of the test set composed of 3.475 samples. On the other hand, we can rely on research in the field of panoptic-segmentation regarding this dataset, and utilise a model trained on the train set of Cityscapes to segment the test and train-extra, as well as the DriveU dataset for more samples. Thus, some further argument for Cityscapes is that both, the DriveU and the Cityscapes dataset have the same image resolution (2084×1024 pixels), very similar camera positioning and are both recorded in Germany. The latter matters because the lighting, brightness and also the colour spectrum are dependent on the position of the sun, humidity, temperature and reflected light of the terrain or briefly the geolocation and weather. With this conformance of the two datasets, it is easier to transfer image classification models between them as it is our plan, with the traffic light classifier, and the panoptic-segmentation model. One sample from the Cityscapes dataset is depicted in fig. 4. For further information see the paper about the dataset [22].

4.3 DriveUC

We want to train our traffic light classifier on the data from DriveU and deploy it on Cityscapes. However, for obtaining decent results we need the distributions of traffic light images in DriveU and Cityscapes to be as close as possible, thus our model can generalise from one dataset to the other. As we work with only the cropped images of traffic lights, computational efficiency demands the creation of a dataset that does not contain the full 2048×1025 images but only the crops with much smaller pixel measures. This means we have to make a very important decision being what to crop, since with all further data augmentations we are limited to these crops. The first idea would be to just use the very accurate bounding boxes that come with the DriveU ground truth. Doing so would provide us with a dataset of perfectly cut out traffic lights with almost no background, looking like fig. 5. This would not be an issue if the Cityscapes samples looked alike, but they do not. In the Cityscapes dataset, the bounding boxes of the traffic light containing only one object with almost no background are not given but therefore an instance segmentation. This

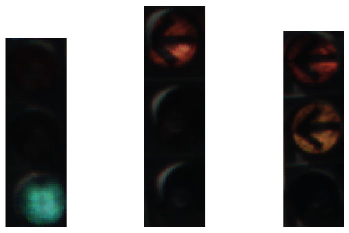


Fig. 5: Some cropped traffic lights from the DriveU dataset.

means we have a traffic light polygon that often does include mounting and housing parts and parts of the pole the traffic light is attached to. Hence, the traffic light is often not in the centre of the image and we are confronted with a significant amount of background pixels. We even raise the number of background pixels further, by cropping not only the polygon but a rectangle including the polygon, as we can not input the plain instance-segmentation to our model. Another problem is that one polygon instance sometimes includes not only one but a conglomerate of traffic lights if they are very close to each other (fig. 4).

Thus to account for the background pixels leaking into the polygon threw bad annotations and/or the annotation of the housing/pole, additional traffic lights and mounting parts we perturb the accurate bounding boxes of DriveU in the cropping process. Doing so, we hope to achieve an assimilation of the traffic light crops of Cityscapes and DriveU.

Now the concept of the cropping scheme is, that when a traffic light sample in DriveU is large enough (we decided for five pixels in width and five pixels in height), we do not only crop the original bounding box but also three additional bounding boxes with randomly perturbed bounds.

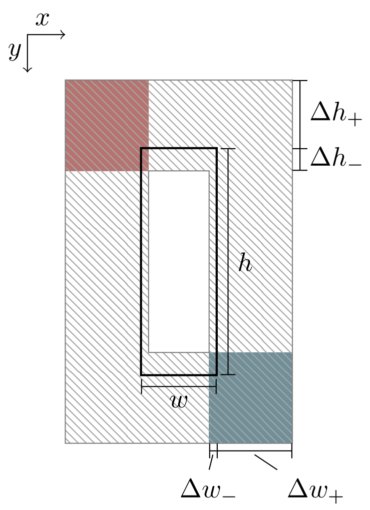


Fig. 6: A visualisation of how the random cropping process works. The original bounding box is depicted in black. The maximum and minimum height and width borders, which include all possible random crops, are shown by the gray area. For a new bounding box the left upper corner is sampled from a discrete uniform distribution limited by the red area and the lower right corner by one limited by the blue area. The ratios in the sketch are accurate with respect to eq. 4.1 and eq. 4.2.

For this we define four scaling factors, namely:

$$s_{h_+} = 0.3, \quad s_{h_-} = 0.1, \quad s_{w_+} = 1, \quad s_{w_-} = 0.1. \tag{4.1}$$

And then, using this scaling factors, we construct the boundaries for every sample (with width w and height h) individually by:

$$\Delta h_+ = s_{h_+} \cdot h, \quad \Delta h_- = s_{h_-} \cdot h, \quad \Delta w_+ = s_{w_+} \cdot w, \quad \Delta w_- = s_{w_-} \cdot w. \tag{4.2}$$

The resulting values are finally rounded to integers (i.e. pixel values). If (x_0, y_0) and (x_1, y_1) are the upper left and the lower right corners of our original bounding box in the coordinates of the 2048×1024 pixel image, then we will obtain the coordinates of our perturbed bounding box by:

$$\tilde{x}_0 \sim \mathcal{U}(x_0 - \Delta h_+,\ x_0 + \Delta h_-;\ 1) \tag{4.3}$$

$$\tilde{y}_0 \sim \mathcal{U}(y_0 - \Delta w_+,\ y_0 + \Delta w_-;\ 1) \tag{4.4}$$

$$\tilde{x}_1 \sim \mathcal{U}(x_1 - \Delta h_-,\ x_1 + \Delta h_+;\ 1) \tag{4.5}$$

$$\tilde{y}_1 \sim \mathcal{U}(y_1 - \Delta w_-,\ y_1 + \Delta w_+;\ 1). \tag{4.6}$$

In this context $\mathcal{U}(a, b; c)$ is the discrete uniform distribution is giving equal probability to all values in $\{a, a + c, \ldots, b - c, b\}$ and zero probability to all other values. One should also note that the coordinate system of an original DriveU image has its origin at the upper left corner with the y values increasing from top to bottom. The principles of this random cropping are shown in fig. 6.

As we already mentioned, we first crop the original image and then, when the traffic light object satisfies the condition of $w > 4$ and $h > 4$, we crop three more images with the procedure described above. If the points defining the bounding box lie outside the picture, we correct them by substituting the x and/or y components that violate the image boundaries with the respective boundary value. To prevent sampling the same image twice, we discard all crops that have already been sampled and sample again until we get a crop that has not been in our collection already. We refer to the dataset we obtain from this procedure as DriveUC.

Comparing the traffic light features of DriveU, DriveUC and Cityscapes[4], shown in fig. 23 under sec. C in the appendix, one notices that with the perturbed cropping process we can make the Cityscapes and the DriveU traffic light samples look more alike, regarding depth, width and height. Especially the more symmetric histograms of width and height, with a longer tail towards bigger traffic light crops, are a common property of Cityscapes and DriveUC, while DriveU shows no strong distinction of this property. One notices, however, that the bump in the width histogram of Cityscapes could not be reproduced in DriveUC and that the tail of the width and height histogram, we talked about, is still more developed in Cityscapes than in DriveUC.

Another thing, notable when looking at the feature-histograms (fig. 23), is the similarity between the depth histograms of DriveU and DriveUC. The depth values are calculated as the median of the depth of all pixels in the crop, where the median is chosen because of its more stable properties when it comes to noisy data. And indeed, even with the crops of DriveUC involving more background pixels, the depth of the traffic lights seems not to shift significantly, when perturbed-cropping is applied. Only at further depths, the values start to differ between DriveU and DriveUC, since the crops tend to be smaller and hence even the median becomes more inaccurate.

In tab. 14 under sec. B in the appendix, we summarised the appearances of the different classes for the

[4]For the train-extra and the test set of Cityscapes we use our panoptic-segmentation produced optained with the model from sec. 6, while for the train set we use the ground truth. The bounding boxes are constructed by forming the smallest rectangle enclosing all pixels of an instance in the segmentation.

newly created DriveUC dataset, together with the mean and standard deviation of with and height in every class.

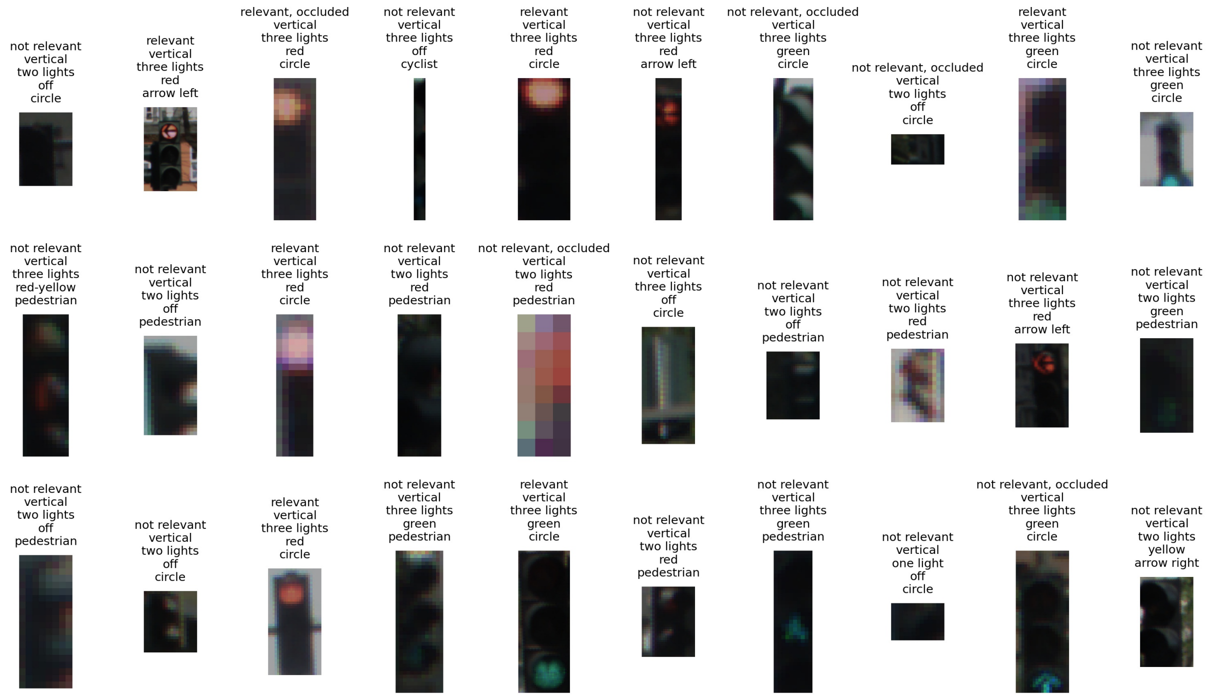
5 Traffic Light Classification

In this section we want to discuss the development and training of the state, type and relevance classifier. It consists of a convolutional neural network which determines the state and the type and a simple feed-forward neural network that predicts the relevance. The first network uses the cropped images from DriveUC while the second one makes its prediction based on the state, type, the position relative to the camera (i.e. to the car) and the pixel measures of the sample. The networks are separately trained, on DriveUC. After the training, they are chained together to the complete model, with the type and state information in the input of the relevance classifier being produced by the first network. Finally, this combined classifier is tested on the test set of DriveUC to obtain an estimate of its performance.

5.1 Type and State Classification

5.1.1 Analysing the Data

We first want to take a look at the data we are working with. Concerning the type and state classification task, the data comes as cropped traffic light images from the DriveUC dataset, with some samples depicted in fig. 7. The samples were drawn at random to give a decent impression of the composition of the data.



not relevant vertical two lights off circle
relevant vertical three lights red arrow left
relevant, occluded vertical three lights red circle
not relevant vertical three lights off cyclist
relevant vertical three lights red circle
not relevant vertical three lights red arrow left
not relevant, occluded vertical three lights green circle
not relevant, occluded vertical two lights off circle
relevant vertical three lights green circle
not relevant vertical three lights green circle
not relevant vertical three lights red-yellow pedestrian
not relevant vertical two lights off pedestrian
relevant vertical three lights red circle
not relevant vertical two lights red pedestrian
not relevant, occluded vertical two lights red pedestrian
not relevant vertical three lights off circle
not relevant vertical two lights off pedestrian
not relevant vertical two lights red pedestrian
not relevant vertical three lights red arrow left
not relevant vertical two lights green pedestrian
not relevant vertical two lights off pedestrian
not relevant vertical two lights off circle
relevant vertical three lights red circle
not relevant vertical three lights green pedestrian
relevant vertical three lights green circle
not relevant vertical two lights red pedestrian
not relevant vertical three lights green pedestrian
not relevant vertical one light off circle
not relevant, occluded vertical three lights green circle
not relevant vertical two lights yellow arrow right

Fig. 7: Some cropped traffic lights together with their label.

For our purpose, it is sufficient to only use the state- and the type-information and set aside all the other classes. The type is taken into account because it contains information about who the traffic light is important for, which can later be supplementary when classifying the relevance. The relevance feature, on the other hand, is not included in this first model, due to its contextual nature not contained in the images. Meaning, if even a human is not able to classify the relevance feature from the given information than the network is also very unlikely able to. And for the samples in fig. 7, as well as the rest of DriveUC, it is almost impossible to guess the relevance feature without any further context.

However, adding a class that can only hardly be derived from the input data introduces instability in the learning process, since the weights will also be adjusted to minimise the loss of this class. Therefore, we leave out the relevancy feature here and come back to it later in sec. 5.2. This leaves us with a two vs. rest multiclass-classification problem with a label of length twelve, consisting of five entries for the state ({off, red, red-yellow, yellow, green}) and seven entries for the type ({circle, arrow straight, arrow left, arrow-straight left, arrow right, pedestrian, cyclist}).

Besides omitting the relevance class, we also discard some samples from the dataset, namely the ones that have a pixel width below three. The reason for this is that for our analysis in sec. 7, we are mainly interested in the traffic lights that are closer to the vehicle and hence wider and taller. First of all, they are easier to classify as they provide more pixel-information and second, the pedestrian crossings they are possibly related to, are more likely to be fully depicted or even existent in the image.

If we now apply this minimum width threshold we reduce the DriveUC dataset from 791.547 to 777.918 samples, discarding about 1.7 % of the dataset. The exclusion of those samples is, however, quite beneficial to the learning process, since samples below a width of three often do not contain any information at all (see fig. 7, first row, fourth picture) while the labelling was probably done with contextual information. Using such samples can again result in instability during training and for them merely being of interest to us, nor make up a major part of our dataset, we exclude them from training and testing.

In this chapter (sec. 5.1) when we talk about DriveUC and no further specification is given, we mean this reduced dataset.

5.1.2 The Model

The used model is the convolutional neural network proposed in [14] by the name Dense Convolutional Network (DenseNet). It is a widely used network in image classification [43] and although it was developed in 2016 the vanilla implementations still ranks 22th on CIFAR-10, 16th on CIFAR-100 and 111th on ImageNet [36] according to [44]–[46], using top-1-accuracy as criterion.

The network implements the idea of a direct connection from one layer to all subsequent layers. Hence, with $H_i(\cdot)$ denoting a composite function of for example batch normalisation and convolution followed by a rectified linear unit, the output of the i^{th} layer is defined as:

$$\boldsymbol{x}_i = H_i([\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{i-1}]), \qquad [\cdot] \text{ denotes tensor concatenation.} \tag{5.1}$$

This architecture increases information and gradient flow threw the network by its dense connectivity and also facilitates the combination of low and high level features of which the first are gained closer to the input-layer while the latter are extracted deeper in the network [2, p. 331].

The architecture we use is the DenseNet-B implementation for which the composite function consists of the following consecutive operations:

$$\text{bn} \;\rightarrow\; 1 \times 1 \text{ conv} \;\rightarrow\; \text{ReLU} \;\rightarrow\; \text{bn} \;\rightarrow\; 3 \times 3 \text{ conv} \;\rightarrow\; \text{ReLU}. \tag{5.2}$$

The letter B is derived from the term bottleneck and aims at the first three parts of the upper flow diagram that reduce the number of input feature maps for the succeeding functions. If necessary, a stride of one and zero-padded edges are used before the conv-operations, to keep the feature map dimensions fixed. We do stick with the notation of the paper and call eq. 5.2 a dense layer.
To implement the down-sampling which is a essential part of convolutional-neural-networks the dense layers have to be grouped into dense blocks, which are interconnected with transition layers. The reason for this is that the down-sampling (e.g. pooling) changes the size of the feature maps resulting in the concatenation of layer-outputs in eq. 5.1 no longer being possible. Hence the network has to be divided into several blocks between which the down-sampling of the feature maps is realised by layers build from a 1×1 convolution followed by a 2×2 average pooling. Together they cut in half the number of feature maps as well as their width and height respectively. After every dense block there is also the opportunity to implement a dropout function, that randomly drops features during training imposing some regularisation on the network by preventing co-adaptations of the weights to the training data.
With the knowledge about this elementary building blocks the model is now defined by its initial layer,

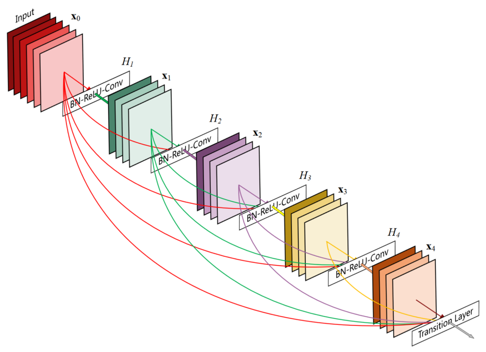


Fig. 8: A visualisation of a 5-layer dense block with a growth rate of $k = 4$. For clarity reasons the bottleneck layers have been omitted. The graphic is taken from [14].

its dense blocks with the dense layers they contain respectively, the classification layer and the growth rate k of the network. Hereby growth rate is a hyperparameter that sets the number of feature maps the conv-operations in eq. 5.2 produce. The bottleneck-convolution outputs $4k$ feature maps, while the 3×3 operation outputs k. This results in k outputs for every dense layer which makes a total of $k_0 + i \cdot k$ feature maps after the i^{th} layer of a dense block if the initial number of channels is k_0. An conceptual example of a dense block is visualised in fig. 8.
The final implementation of the model is shown in tab. 1. As said above, we limit ourselves to the state- and type-classification in this part, thus the number of outputs is $n = 12$, of which the first five, as well as the last seven entries, are mutually exclusive. For this reason, we apply the softmax-operation separately to those two domains and not to the whole label. One also could have used thirty-five labels and perform a one vs. all classification, however that would have drastically reduced instances in classes like (red-yellow, arrow-straight), while also entangling the state and type classification, which is against the nature of the problem itself, as the type information should be completely detached from the state

Layer	Configuration
Initial Layer	$\begin{bmatrix} 7 \times 7 \text{ conv, stride } 2 \\ 3 \times 3 \text{ max pool, stride } 2 \end{bmatrix}$
Dense Block (1)	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 2 \times 2 \text{ average pool, stride } 2 \end{bmatrix}$
Dense Block (2)	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 2 \times 2 \text{ average pool, stride } 2 \end{bmatrix}$
Dense Block (3)	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$
Transition Layer (3)	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 2 \times 2 \text{ average pool, stride } 2 \end{bmatrix}$
Dense Block (4)	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$
Classifiacation Layer	$\begin{bmatrix} 7 \times 7 \text{ adaptive average pool} \\ n\text{-D fully connected layer, softmax} \end{bmatrix}$

Tab. 1: The used DenseNet architecture. Here the conv-operations corresponds to the sequence bn-conv-ReLU (conv with stride one) and n is the number of classes. The drop-out after every dense block is omitted in the table, since it is only used during training and no substantial part of the model.

and vice versa.

5.1.3 Preprocessing of the Data

Before we can pass our samples to the network, we first have to resize them to equal width and height. To keep the rough ratio of 3:1 (see tab. 14) and still have the image dimensions being powers of two, we choose an image size of 128×32. The reason for us wanting divisibility by two stems from the characteristics of the downsampling in the network, wherein every step the feature-map dimensions are cut in half.
To synthetically inflate the number of samples and reduce the likelihood of overfitting there are several possibilities of data augmentation. We give here a list (tab. 2) of some common and easy to implement practises and reason about whether to use them for our purpose and in what way.
All the augmentations are applied at random, meaning that for every sample the augmentations are different in every epoch. We only specify the domains for the translation, rotation, scaling and brightness, saturation, contrast and hue and then sample the adjustment values uniformly from those.

Augmentation	Description	Applicability	Domains
Affine Transformation	A translation, rotation and scaling of the image.	This operation is useful with reasonably small values for all the operations since we do not want the image to get flipped sideways, turned over, or an active light bulb completely shifted out of the image.	rotation: $(-5°, 5°)$ translation (relative): $x : (-0.1, 0.1)$, $y : (-0.1, 0.1)$ scaling (relative): $(0.8, 1.2)$
Flipping	Flipping the image horizontally or vertically.	We can neither use the horizontal flip nor the vertical flip, as traffic lights contain spacial information that would be changed threw these alterations (e.g. flipping of an arrow pictogram from left to right, flipping the order of the traffic light bulbs putting green on top and red at the bottom).	
Colour Jitter	Changing the brightness, saturation, contrast and hue of the image.	As with the affine transformations, we can use this augmentation, but should be careful with the magnitude of the change. Especially with the adjustment of the hue, because a too great of a change here could result in a red traffic light turning green.	brightness: $(-0.3, 0.3)$ contrast: $(-0.3, 0.3)$ saturation: $(-0.3, 0.3)$ hue: $(-0.05, 0.05)$
Crop	Crop out one or more frames from an image.	Since we already cropped the images in DriveUC and further operations of this type could result in the exclusion of the light bulb that is on, we omit this transformation in the training process.	
Gaussian Noise	Sample i.i.d. noise of a Gaussian distribution with mean zero and a global standard deviation and add this noise to the pixel values.	There should appear no problems using this practice if the standard deviation is kept in a reasonable range.	standard deviation: $\sigma = 0.015$

Tab. 2: Selecting the data augmentations. The values for the applied augmentations are sampled from a uniform distribution over the given domains. For the Gaussian noise, the standard deviation is fixed to about 10 % of the standard deviation of the colour channels (eq. 5.3) after the normalisation. Except for the Gaussian noise alteration, which we implemented ourselves, the alterations all come together with the PyTorch library [13].

After the resizing and the application of the alterations, except for the adding of Gaussian noise which follows after, the inputs are normalised in every colour channel, to the respective mean and standard deviation of the DriveUC dataset:

$$\mu = (0.181, 0.182, 0.184), \qquad \sigma = (0.166, 0.158, 0.158)^{5} \tag{5.3}$$

This is done to facilitate the back-propagation during training an accelerate computations [47] as well as balancing the features to similar ranges.

5.1.4 Experiments

As our criterion, we use the cross-entropy with weighted classes and as an optimiser, we chose the Adam algorithm proposed in [7] and briefly discussed in sec. 2.2.3. Weighted classes mean that we weigh the outputs of our model in the loss, by the appearance of the corresponding class:

$$w_c = 1 - \frac{n_c}{n_{all}}, \qquad n_{all} = \sum_{c \in \mathcal{C}} n_c. \tag{5.4}$$

In this equation, n_c denotes the number of appearances of the class c in the dataset used for training. The weighing as in eq. 5.4 is done for the type and state individually and using this weights we obtain the loss of a mini-batch of size r as:

$$\mathcal{L}_\lambda(\boldsymbol{\theta}) = -\frac{1}{r}\Big(\sum_{k=0}^{r}\sum_{t\in\mathcal{T}}\sum_{s\in\mathcal{S}} w_t\, \mathbb{1}\{\boldsymbol{y}_k^{\text{type}} = t\}\, \log(o_{k,t}) + w_s\, \mathbb{1}\{\boldsymbol{y}_k^{\text{state}} = s\}\, \log(o_{k,s})\Big) + \lambda\|\boldsymbol{\theta}\|_2. \tag{5.5}$$

Here

$$o_{k,c} = (\mathcal{N}(\boldsymbol{x}_k, \theta))_c \tag{5.6}$$

is the entry of the models output of the k^{th} sample, corresponding to the class c. The $\boldsymbol{y}_k$ is either the states or type ground truth of the k^{th} sample, whereby the correspondence is indicated with upper case letters. Finally $\mathbb{1}$ is the indicator function, being one if the condition in its brackets is full filled and zero otherwise and $\mathcal{T}$, $\mathcal{C}$ are the sets of possible type and state classes respectively.
We set the hyperparameters of the Adam optimiser to the default values ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$) recommended by the paper [7] and use a batch size of 256, leaving us with the decision about the drop rate, the learning rate and the weight decay.
Regarding the simplicity of our classification model, approaches like Bayesian hyperparameter optimisation, proposed in [48], seem too sophisticated and not worth the effort. On the other hand, a grid search [47] is quite simple to implement but computationally expensive. We, therefore, decide for a random search as suggested in [49], which is straight forward to implement and has a reasonable computational cost. We do however apply some small modifications to the algorithm.

[5] Listed are the values that were calculated after the resizing process, however, the networks were trained with $\mu = (0.158, 0.156, 0.161)$, $\sigma = (0.166, 0.158, 0.158)$, is the mean and standard deviation before the resizing. The error had not been noticed until there was no time left for a correction. For the discrepancy not being that big, this should, however, not be a severe mistake.

We start with a random search of the discrete hyperparameter space using the following grid:

$$\text{drop rate} = \{0,\ 0.1,\ 0.2,\ 0.3,\ 0.4,\ 0.5\}, \tag{5.7}$$

$$\text{learning rate} = \{10^{-n} \mid n \in [\![0,\ 5]\!]\}, \tag{5.8}$$

$$\text{weight decay} = \{10^{-n} \mid n \in [\![0,\ 5]\!]\}. \tag{5.9}$$

For twenty models we uniformly sample drop rate, learning rate, weight decay from the above sets and then train all these models on a small arbitrary subset of the DriveUC training set (roughly 18.000 training, 2.000 validation samples). After the random search is finished we use the *PathSearch* algorithm (alg. 1) on the best parameter configuration regarding the validation loss, to explore the area around them for even better parameters. As alteration schemes we use:

$$\mathcal{P} = \{-0.1, 0, 0.1\} \times \{0.1, 1, 10\} \times \{0.1, 1, 10\} \backslash \{(0, 1, 1)\}. \tag{5.10}$$

Here $\times$ denotes the cartesian product and $\backslash$ the set-theoretic difference (we exclude the identity alteration). The first dimension corresponds to the drop rate, the second to the learning rate and the third to the weight decay. If we now have a parameter vector, say $p = (dr, lr, wd)$ and we alter it with $p = (-0.1, 1, 10)$ then the new parameter configuration produced by APPLY is:

$$\textsc{apply}\,((dr, lr, wd), (-0.1, 1, 10)) = (dr - 0.1,\ lr \cdot 1,\ wd \cdot 10). \tag{5.11}$$

Hence the drop rate (dr) is shifted while the learning rate (lr) and the weight decay (wd) are scaled. alg. 1 does however not take into account that the drop rate is bound to the range $[0, 1]$, which is something that is taken care of in the actual implementation by just discarding branches of the search where the drop rate would become negative or greater one.

Threw out the random- and path-search we also use an early stopping mechanism that stops training if the validation loss shows strong and rapid fluctuations or is even increasing over a period of epochs. For this we use the *earlyStopping* algorithm (alg. 2). This algorithm applies a mean-filter-convolution to the validation loss and then performs a linear regression on a defined interval. If this regression results in a slope above zero, we stop the training. We use *earlyStopping* with a filter size of five and train for ten epochs before checking the early stopping condition. The interval size is set to fifty, meaning that after epoch $t \in \mathbb{N}_0$, we mean-filter the loss and then perform a linear regression either on all validation loss data points, if $t \leq 50$ or the most recent fifty data points if $t > 50$. We limit the total number of training epochs with 150. By this, together with the early stopping and the reduction of the dataset, we can drastically reduce the running time of the search, while the relative results between models of different hyperparameters should not change significantly when using the whole dataset and train to a higher number of epochs.

When finished with this combination of random- and path-search we chose the best hyperparameters on the base of the validation f-scores and the loss. We then train a model with the respective hyperparameters for 500 epochs on the whole DriveUC train set. From this 500 epochs, we then chose the model with the minimal loss on the validation set (10 % of the train set) and train it again for 500 epochs, however with a learning rate reduced by a factor of 10. Finally, we chose the model with the highest validation f-score from this last training to use it in our traffic light classification pipeline. The f-score is the harmonic mean of the precision and recall[6], both being the predominant performance measures in the context of traffic light classification and recognition [20]. From now on, if we are dealing with a multiclass classification problem and no further information is given, the f-score, precision and recall, always denote the sample weighted mean over the respective score in every individual class.

5.1.5 Results

The results of the random path search are listed in tab. 15 under sec. B in the appendix. A visualisation of the weighted validation f-score for the searched configuration is given in fig. 9. We notice that the drop

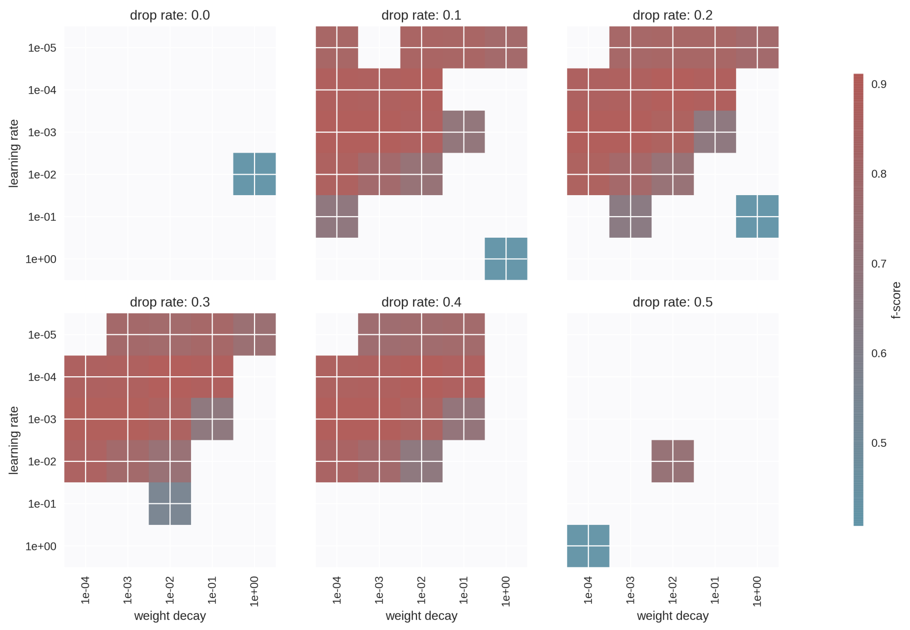


Fig. 9: The random path search visualised. Every patch (i.e. cross) corresponds to a certain configuration of drop rate, learning rate and weight decay. The colour of the patch indicates the best f-score on the respective configuration. The f-score is calculated by taking the weighted mean overall class individual f-scores. The white patches correspond to configurations that were not explored.

rates of 0 and 0.5 were merely searched, while the excessive search took place at drop rates of 0.1, 0.2, 0.3, 0.4. It also seems that the drop rate does not have a severe impact on the performance of the network. We can see this when looking at the patches of the drop rates that were searched more comprehensively, with only a slight difference in the f-score for every learning rate and weight decay patch going to higher drop rates. Another thing that we notice from the plot, is that the mesh size of our random-path-search grid is quite reasonable since neighbouring patches (i.e. hyperparameter configurations) do not show a drastic change in validation f-score around our optimum, which means, that even with a finer grid there

[6]Performance and recall are introduced in [20], where also their fit for the traffic light classification and recognition task is discussed.

should not be much improvement in performance. This assumption is however only true if the loss surface in the hyperparameter space is continuous and does not show a rapid increase or decrease on distances below our mesh size.
Since there is a discrepancy between the hyperparameters producing the best validation loss, and the ones producing the best validation f-score (tab. 15), we chose the model that performs best regarding its position in the validation f-score and loss ranking. The model with:

$$\text{drop rate} = 0.3, \tag{5.12}$$

$$\text{learning rate} = 1 \times 10^{-4}, \tag{5.13}$$

$$\text{weight decay} = 1 \times 10^{-2} \tag{5.14}$$

ranks first in the f-score category and fifth in the loss category, while the leader in the loss category only ranks ninth in the f-score category. Hence we chose the hyperparameters of the leader of the f-score category for our final model. The final choice should not make a big difference, however, since around this optimum the performance of the models is quite similar (fig. 9), and the fluctuation could be very well caused by the initialisation and the parsing order of the dataset during training.
The course of the stage wise training of the type and state classifier is depicted in fig. 10. We reach the best validation f-score in epoch 794 with the results summarised in tab. 3. At last we test the model on

Category	Precision	Recall	F-Score	Accuracy
total	0.959829	0.959272	0.959435	0.959272
state	0.973826	0.973814	0.973798	0.973814
type	0.945832	0.944729	0.945073	0.944729

Tab. 3: The performance of the chosen type and state classifier on the validation set. The precision, recall and f-score are calculated by taking the sample weighted mean over all the respective classes.

the DriveUC test set obtaining the results in tab. 4. The results for every class individually can be found in tab. 16 in the appendix under sec. B. The results listed there are the ones for the combined classifier, including the relevance as well, however, with the type and state classifier working unchanged, they are the same as the ones obtained from this testing. We notice that besides the arrow-straight-left class,

Category	Precision	Recall	F-Score	Accuracy
total	0.907654	0.906522	0.906535	0.906522
state	0.940041	0.939172	0.939251	0.939172
type	0.875266	0.873871	0.873818	0.873871

Tab. 4: The performance of the chosen type and state classifier on the validation set. The precision, recall and f-score are calculated by taking the sample weighted mean over all the respective classes.

which only containing 42 training samples and 8 test samples, while also being quite similar to the arrow

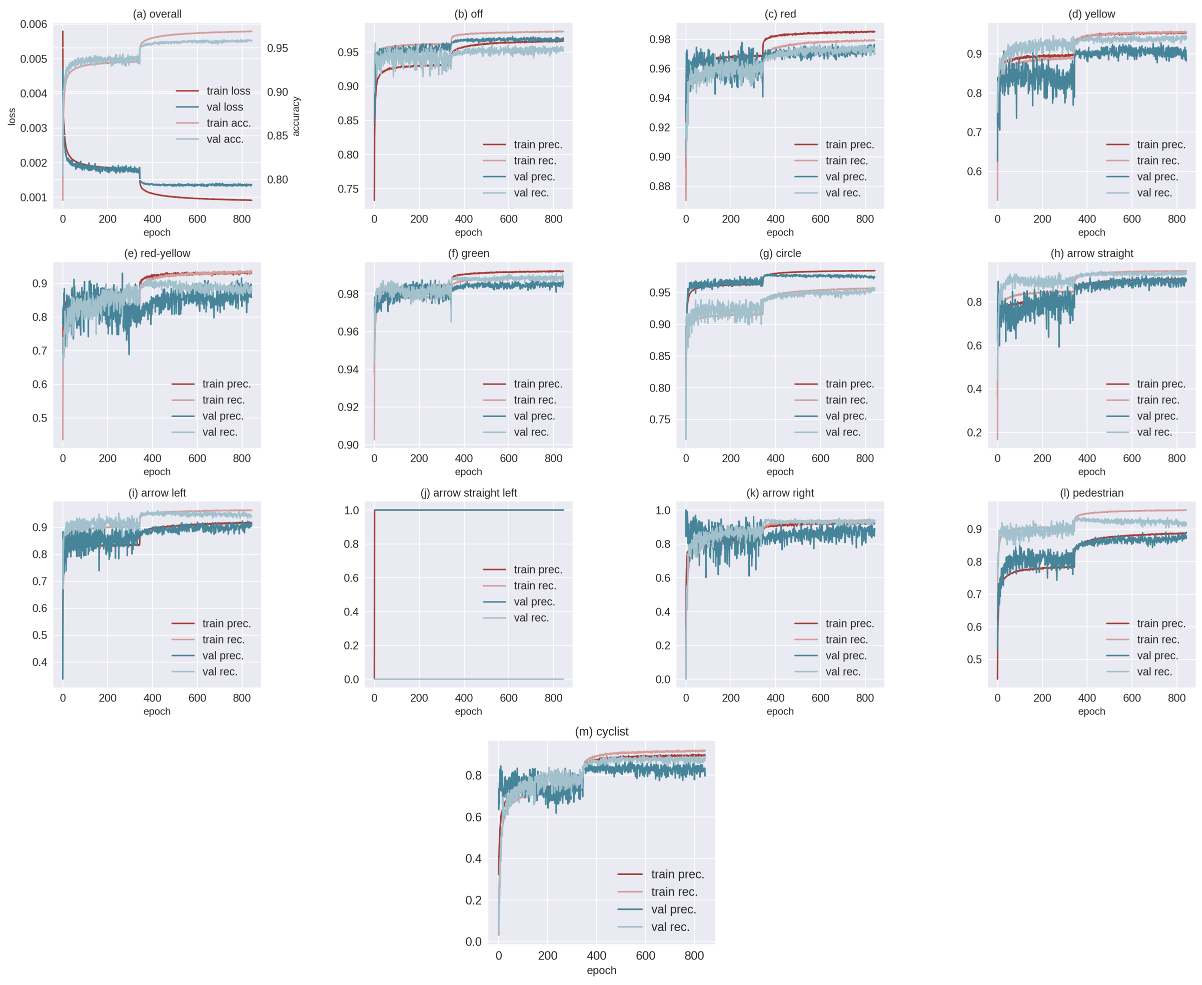


Fig. 10: The scores for the stagewise training of the type and state classifier. We plotted the accuracy and the loss for all classes together, and recall and precision for every class individually.

straight and arrow left class, the classifier performs quite well with an overall sample weighted f-score of 0.9065.

5.2 Relevance Classification

5.2.1 Analysing the Data

The data we want to give to our relevance classifier looks as follows:

$$d_i = (x_i,\, y_i,\, width_i,\, height_i,\, depth_i, state_i,\, type_i), \qquad i \in [\![0, n]\!]. \tag{5.15}$$

The given feature vector is build from the i^{th} sample in a dataset of a total of n samples with the values of x_i, y_i, $width_i$, $height_i$ given in pixels, while the $depth_i$ is given in meters. x_i, y_i define the centre of the bounding box rectangle and the type and state are one-hot-encoded.

The reason for the choice of units is that we later want to use our developed model on Cityscapes. As discussed in sec. 4 the images of both datasets are similar concerning the relative point of view and image measures. However, the disparity does not match, since the focal length (f) of the cameras and their baseline (B, i.e. distance of the objectives) are different in both cases. Hence, we can not use the raw disparity data but have to calculate the depth. This is done by using the formula:

$$depth = \frac{f \cdot B}{disparity}. \tag{5.16}$$

Due to the accuracy of the depth information, it seems quite reasonable to assign only a depth scalar to a traffic light. Even with a higher depth resolution the use of more then a scalar for the distance should have no significant impact, concerning that information about the relevance feature is more likely to be encoded on a scale of meters and not on the scale of the housing parts of a traffic light.
The idea behind obtaining this scalar is straight forward, as we just calculate the median disparity in the bounding box of a sample and then use eq. 5.16 to obtain the depth. We do not use the mean, since it is more vulnerable to noisy data compared to the median, as we have already seen in sec. 4.3 with fig. 23.
In fig. 11 the continuous features of eq. 5.15 are visualised on the complete DriveUC dataset. We do realise some structure in the data, proving our intuition right. We would intuitively believe, that relevant traffic lights are more likely to be closer to the car and have therefore not only a smaller depth but also appear bigger in the pictures (i.e. greater width and height). For the depth of the irrelevant class, we would expect a long tail in the distribution since irrelevant traffic lights can practically appear at every distance while the relevant distribution should not have this tale. We also would expect the traffic lights at every x-position in the picture, however, there should not be any at the lower y-positions since traffic lights are positioned at heights of above three meters.
Although the data is consistent with our expectations, the task of classifying it appears to be a hard one. This is because the marginal distributions do heavily overlap as one can see in fig. 11. With this being the case we certainly have to utilise models that learn joint distributions of the data to obtain a reasonable result. Since we do not want to invest much work in the pre-processing of the data we go with a neural network, as it can learn arbitrary decision boundaries and is not limited to certain shapes for them.

5.2.2 The Model

For this classification task, we use a simple feed-forward neural network as described in sec. 2.2.2 with ReLU as activation function after each layer (except the output layer) and dropout as well as batch normalisation before each hidden layer. Since we are dealing with a binary classification task (i.e. relevant/not relevant) we decide for only one output and therefor equip the classification layer with a sigmoid function.
As we also want to explore different hidden layer configurations we define two hyperparameters that finalise the architecture of our model, namely depth and maximum width.
The maximum width is motivated by research that shows, that it is better to have deeper and rather narrow instead of shallow networks if the goal is to learn feature spaces of advanced complexity [50], [51].
From this hyperparameters we obtain our layer configuration by setting the output layer to size one and

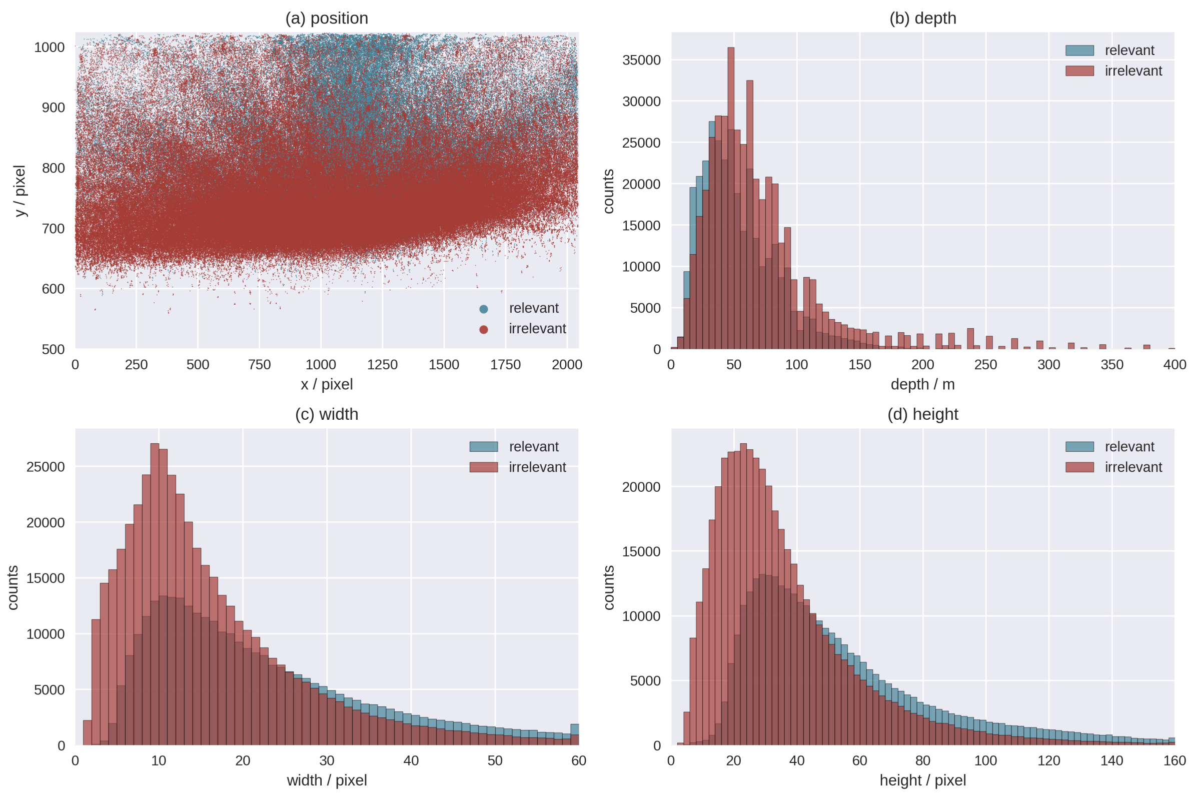


Fig. 11: Visualisation of the continuous features from DriveUC. (a) shows the x_i- and y_i-position of the traffic lights as a scatter plot, whereby the y-axis starts at 500 px. (b) shows the *depth* of the samples in meters, and (c) and (d) show the width and height in pixels respectively. In the histograms some outlier samples are neglected for visualisation.

the input layer to size fifteen[7]. In between, we start to add layers moving backwards from the output increasing the layer size by two in each step until we reach the maximum width or the maximum depth. If we reach the maximum width before the maximum depth we start to successively add layers, with the same size as already existing layers (excluding the input and the output layer) and group layers of alike size. This successive procedure starts by adding another layer of the same size to the first group of hidden layers and then do the same for the next group and so on, moving towards the output layer until we reach the wanted depth. We keep the maximum width fixed with 64 and only vary the depth. As an example we want to give the layer configuration of a network with a depth of ten:

$$\text{layer sizes} = (15,\ 64,\ 64,\ 32,\ 32,\ 16,\ 16,\ 8,\ 4,\ 2,\ 1). \tag{5.17}$$

Finally, before passing the data into the model we normalise the $x_i, y_i, depth_i, width_i$ and $height_i$ to mean zero and standard deviation one so they are all on similar ranges.

[7]The input is fixed by the size of our feature vector (i.e. fifteen) and the output is fixed by the number of our classes (i.e. one).

5.2.3 Experiments

We again use the Adam optimisation algorithm with the same values for the decay rates and the numerical stability value as in sec. 5.1.4. We do also stick with the cross-entropy as the loss function (in this case the binary cross-entropy). This means we have to find the depth of the network, the drop rate, the learning rate and the weight decay. Although the model is far smaller than the model in sec. 5.1 the number of hyperparameters, on the other hand, grew bigger. Therefore we still favour our random search followed by a path search (alg. 1) over the grid search and also use early stopping (alg. 2) again.
As ranges for the random search we use:

$$\text{depth} = \{n \mid n \in [\![6, 16]\!]\}, \tag{5.18}$$

$$\text{drop rate} = \{0,\ 0.1,\ 0.2,\ 0.3,\ 0.4,\ 0.5\}, \tag{5.19}$$

$$\text{learning rate} = \{10^{-n} \mid n \in [\![0,\ 5]\!]\}, \tag{5.20}$$

$$\text{weight decay} = \{10^{-n} \mid n \in [\![0,\ 5]\!]\}, \tag{5.21}$$

and sample twenty configurations uniformly from these ranges. For the random as well as the path search we use the complete DriveUC train set with a train, validation ratio of 0.9 and a batch size of 512. The early stopping interval remains at fifty and the mean filter at size five, the maximum number of trained epochs is 500 and we wait for ten epochs before we start checking the early stopping condition. When finished we chose the best performing model based on the validation loss and start a path search which uses:

$$\mathcal{P} = \{-1, 0, 1\} \times \{-0.1, 0, 0.1\} \times \{0.1, 1, 10\} \times \{0.1, 1, 10\} \backslash \{(0, 0, 1, 1))\} \tag{5.22}$$

as alteration schemes, trying to minimise the validation loss. Here the depth and the drop rate are shifted while the learning rate and weight decay are scaled. Note that the early stopping and the epoch threshold are applied to the path search as well, and that we discard search branches with invalid values for the drop rate (e.g. drop rate $= -0.1$).
When the search is finished, we chose the optimal hyperparameters based on the validation loss and f-score and train a new model for 1.000 epochs with these parameters on the DriveUC train set, using 10 % of the data for validation. When the learning is finished we chose the best performing model, regarding the f-score, test it on the DriveUC test set and then use it in our combined classifier.

5.2.4 Results

The results of the random path search are listed in tab. 15 under sec. B of the appendix. We notice that the optimal validation loss and the optimal validation f-score is not produced by the same model and we, therefore, use the approach that we also used in sec. 5.1, namely selecting the model that performs best in one category, and ranks high in the other. The model with the configuration:

$$\text{depth} = 17, \tag{5.23}$$

$$\text{drop rate} = 0.0, \tag{5.24}$$

$$\text{learning rate} = 1 \times 10^{-3}, \tag{5.25}$$

$$\text{weight decay} = 1 \times 10^{-5} \tag{5.26}$$

ranks first, regarding the f-score and third in the loss category. The contending leader in the loss category only ranks eighth in the f-score ranking, hence, we chose the first model for further training.
In fig. 12 we plotted the course of training for the relevance classifier with the selected hyperparameters. The training progress saturates at about 200 epochs already, with the optimal f-score being reached in epoch 145.
We select the model with this optimal f-score and test it on the DriveUC test set containing 245.003

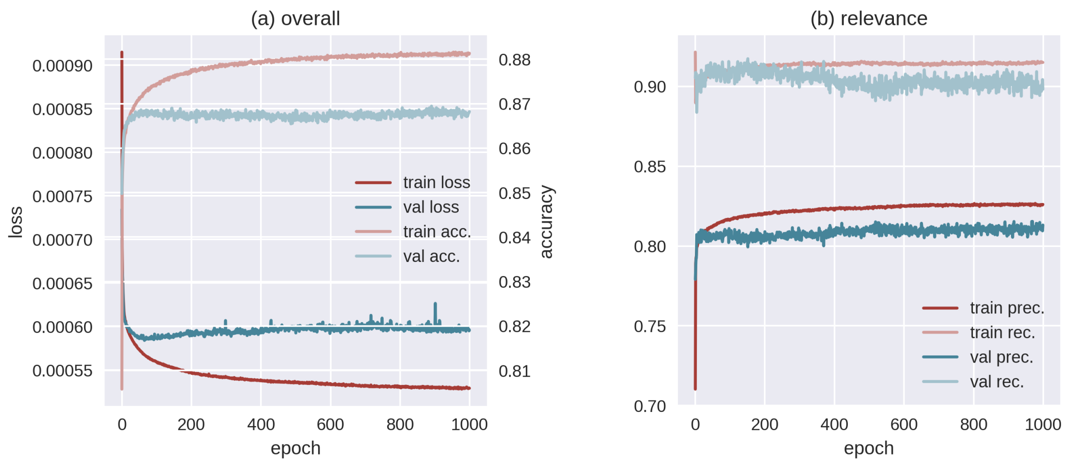


Fig. 12: The f-score, accuracy, precision and recall scores in the course of the relevance classifier training. The batch size was set to 512 and we trained for 1.000 epochs. The highest validation f-score of the model is reached in epoch 145. On the right-hand side we plotted the overall loss and accuracy and the precision and recall for the relevance class.

samples. The results for the validation and test set are summarised in tab. 5. We report an f-score of

Set	Precision	Recall	F-score	Accuracy
validation	0.8059	0.9137	0.8564	0.8690
test	0.7849	0.8996	0.8383	0.8583

Tab. 5: The scores on the validation and test set for the model with the best f-score on the validation set.

0.8383 and a accuracy of 0.8583 on the DriveUC test set. For a binary classification task this is not great, however with 58.57 % irrelevant and 41.43 % relevant in all of DriveUC (tab. 14) the classifier is better than the classifier, which always selects the dominant class in the dataset. The reason for this rather low accuracy is probably the similarity of the relevant and irrelevant class in the features we used (fig. 11), making it difficult for the network to give accurate predictions.

In this context it is also interesting to evaluate the networks calibration, meaning how accurate the probabilities given by the network actually are. Following the procedure described in [52] we obtain the plots in fig. 13. From the diagram, we can see that the network is calibrated quite well, as the

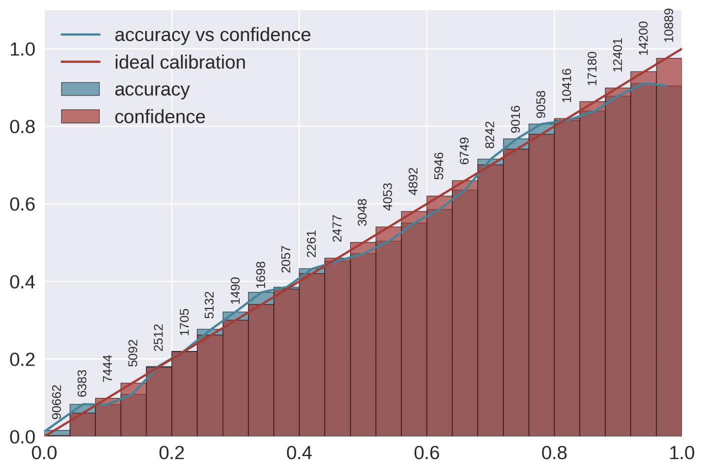


Fig. 13: The calibration of the selected relevance classifier on the DriveUC test set. We wrote the number of samples contained above the respective bin.

accuracy vs. confidence line follows, with few abbreviations, the ideal calibration. We notice, however, that especially at higher confidence/accuracy levels, on the right-hand side of the diagram, there is a larger discrepancy between the model's accuracy and its believes about its accuracy. This can be useful later when discarding samples that the classifier is uncertain about (sec. 7).

5.3 Combined Classification

At last, we now want to study the performance of the classification pipeline build from the combination of the type and state as well as the relevance classifier. In this pipeline the input to the type and state classifier is still an image crop, however, the type and state part of the relevance classifications feature vector eq. 5.15 is now produced by the type and state classifier. We test this pipeline on the DriveUC test set, however, excluding traffic lights below a width of three, similar to sec. 5.1. For the rest of this section (sec. 5.3), when we talk about the DriveUC test set, we mean this reduced dataset. The results for the different categories is listed in tab. 6, and the results for the individual labels is listed in tab. 16 under sec. B in the appendix.

Analysing the results we notice several things. First of all the type and state classification is quite accurate with and f-score of 0.94 for the state and 0.88 for the type. Only the "arrow straight left" label falls out of line, which is however not that surprising, as the whole DriveUC dataset only contains 50 (42 in the train and validation set and 8 in the test set) samples of this class, and the natural tilting of objects in images as well as our rotation in the pre-processing might cause further confusion with the "arrow straight" and "arrow left" labels.

Categorie	Precision	Recall	F-Score	Accuracy
total	0.806622	0.709324	0.742335	0.814600
state	0.940429	0.939573	0.939647	0.939573
type	0.876595	0.875239	0.875163	0.875239
relevance	0.602843	0.313159	0.412195	0.628988

Tab. 6: The results on the three categories of the combined classification on the reduced DriveUC test set. Here we calculated the precision, recall and f-score by taking the sample weighted mean over all the respective classes. The total number of samples in the reduced DriveUC test set is 240.604.

The second matter that is notable is the rapid decrease in accuracy and f-score classifying the relevance, comparing the isolated model tab. 5 to our combined classifier. This means that the relevance classifier is quite reliant on the type and state feature, being lesser of a surprise as the type feature also contains information about the group of traffic participants the traffic light is designated for. Regarding this information, one can already exclude pedestrian and cyclist traffic lights, as they are not relevant for the vehicle. The relationship of the type and relevance, is however not linear, as tab. 7, proves. If it would be a linear relationship, we would expect a positive correlation coefficient, of some greater value, with the coefficients calculated with:

$$Corr(X,Y) = \frac{acc_{XY} - acc_X acc_Y}{\sqrt{(acc_X - acc_X^2)(acc_Y - acc_Y^2)}}, \tag{5.27}$$

where $acc_{X/Y}$ denotes the accuracy of either the X or Y categories and acc_{XY} is the mutual accuracy, where a sample is only classified correct if both, the X and Y classifications are correct.

X and Y	Counts: X and Y correct	Counts: X and Y incorrect	$Corr(X,Y)$
state and type	199679	3632	0.095969
state and relevance	140134	3336	-0.074315
type and relevance	127783	6464	-0.121669

Tab. 7: The correlation between the different categories calculated with the results of the DriveUC test run and eq. 5.27.

6 Panoptic Segmentation

In this section we shortly introduce the panoptic-segmentation pipeline called Panoptic-DeepLab [53], with an modified HRNet-w48 backbone [54]. This pipeline ranks number four on the panoptic-segmentation task on Cityscapes and is the only work providing an openly accessible, pre-trained PyTorch implementation among the best-ranked contenders [55]. A panoptic-segmentation, as proposed in [56], means that the pixel-level semantic segmentation is supplemented by an instance segmentation. This allows us not only to localise point clouds in the image, and their semantic class but also differentiate between two instances that are of the same semantic class. Now with an alike segmentation, we know about the presence as well as the location of individual pedestrians and traffic lights in the image, as well as the surfaces that are road or other ground terrains. Together this is all substantial information for the identification and analysis of pedestrian crossings, and since Cityscapes test, train-extra and the DriveU dataset, do not come equipped with a semantic, or instance segmentation we have to obtain them by utilising the pipeline described in this chapter.

For our purposes, the representation of the outputs is, however, not quite fitting, so following the introduction of the Panoptic-DeepLab pipeline we discuss some postprocessing steps, to get from a panoptic-segmentation to a polygon representation of the contained things and stuff[8] in the image. At the end of this chapter, we discuss the results of the algorithm and the postprocessing on the Cityscapes extra and test, as well as the DriveU dataset.

6.1 Introducing Panoptic-DeepLab with the HRNet+ Backbone

The Panoptic-DeepLab pipeline is built, following a so-called bottom-up structure. This means the pipeline first performs an instance segmentation, which is then followed by the grouping of thing pixels into clusters.

Instead of using the vanilla backbone of Wang et al. [54] for their implementation, Cheng et al. [53] kept

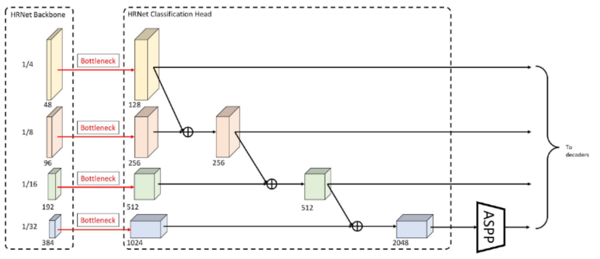


Fig. 14: The HRNet+, proposed in [53], with the kept classification head and the attached ASPP module. The graphic is taken from [53].

the classification head of the HRNet structure after pretraining on ImageNet [36] and also replaced the final average pooling operation and the linear layer with an ASPP module. This setup, called HRNet+,

[8]Things and stuff are commonly used terms in panoptic-segmentation, with a thing being everything that is countable (e.g. person, tree, car, etc.) and stuff is understood as amorphous region of similar material and/or texture (e.g. road, grass, sidewalk).

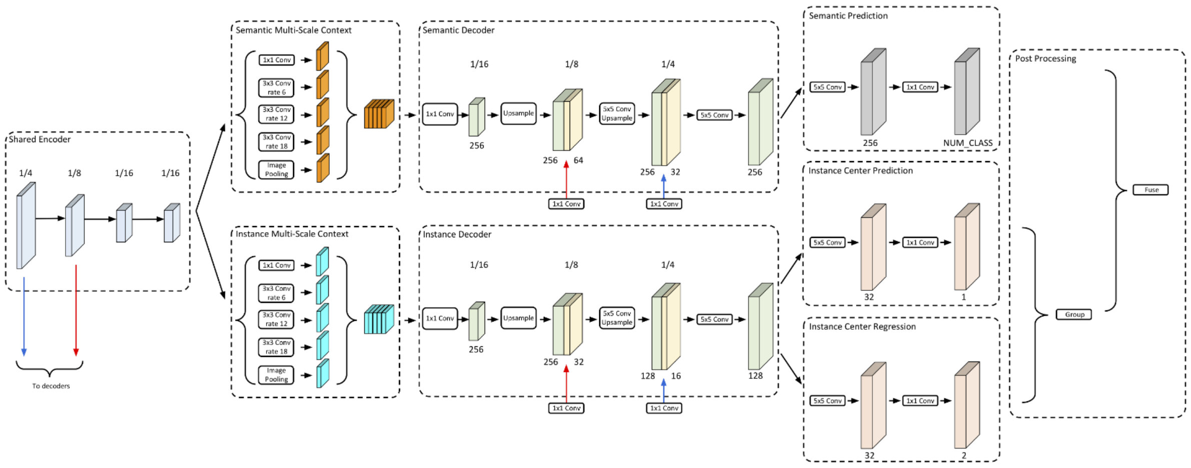


Fig. 15: The Panoptic-Deeplab model. The depicted model is slightly different from the model with the HRNet+ backbone. First of all the encoder and the context modules is replaced with the HRNet+ structure depicted in fig. 14. And seconde, being a result of the different encoder, the produced feature maps of stride 32 have to be adjusted, by adding one more encoder feature map of stride 16 to the decoder, first projecting its channels to 96. From there on the structure of the model follows the depicted scheme with an atrous pooling, the Atrous Spatial Pyramid Pooling (ASPP) and a light-weight decoder, upsampling the sizes of the feature maps, performing one convolution in every stage. Obtaining the instance segmentation is done by predicting the instance centres and regressing every foreground pixel (i.e. pixel of a thing class) to their corresponding centre. In the end, the predicted semantic segmentation and class-agnostic instance segmentation are then fused to generate the final panoptic-segmentation using a "majority vote". The graphic stems from [53].

is then linked to the decoder of the model, depicted in fig. 15. The implementation and pre-trained model that we use can be found at [57], and is indeed a reimplementation of the model described in [53], which is done by the authors of the original papers themselves, however. Another difference between the model provided and the top-ranking model is the additional training data that was used for the latter one.
Now as the last point concerning the Panoptic-Deeplab model, we briefly want to discuss preprocessing. Panoptic-DeepLab was trained and tested only normalising the input to the mean and standard deviation of the respective dataset (i.e. Cityscapes train) and we also stick with this procedure. Hence, also the images of DriveU are fed into the pipeline with no further preprocessing but the normalisation, in this case not to the mean, and standard deviation of Cityscapes but of DriveU.

6.2 The Postprocessing

For the analysis, we conduct in sec. 7, the raw outputs of the panoptic-segmentation are not quite fitting. This is because the outputs come as a 2048×1024 map, assigning every pixel either only a stuff class or a thing class as well as a thing id. Those maps are not only quite memory-intensive and therefore slow to work on, but also do not allow for the easy extraction of properties like neighbourhood and overlap of thing and stuff. Another problem drastically slowing down calculations, when using this point clouds, is the mapping of instances from the image space to the disparity space, needed to make depth estimations of for example pedestrians and traffic lights in the images. For Cityscapes, this transformation is not necessary since the disparity data already comes in image coordinates, but for DriveU this is a major performance issue. To account for our needs and to facilitate performance, we add another post-processing step to the Panoptic-DeepLab pipeline.

First of all, we cut out the area in the image associated with the car, the camera is placed in, since there is arbitrary and highly miss leading segmentation happening in this domain (see fig. 16). Then,

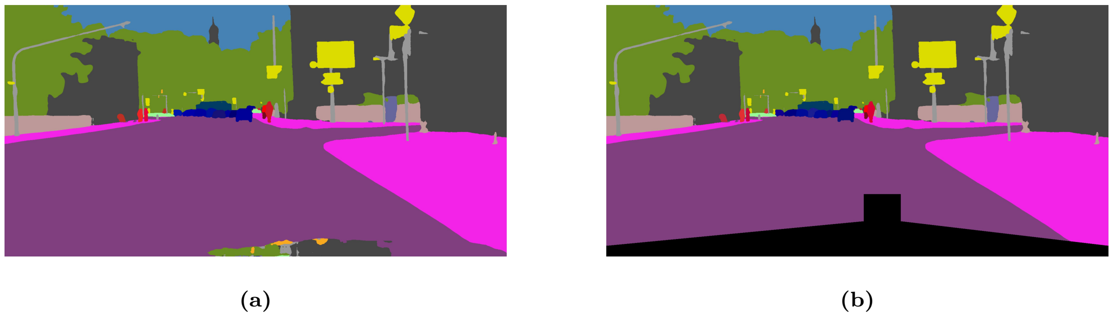
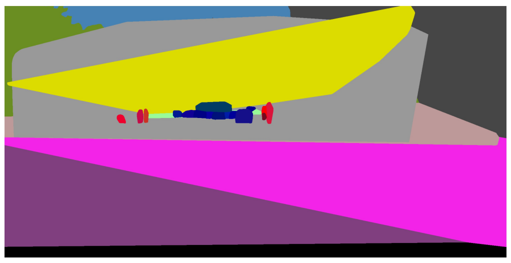


Fig. 16: The image on the left-hand side shows the panoptic-segmentation without a car mask, and on the right-hand side, the car mask is applied. The misclassifications in the domain associated with the car, are quite obvious (a).

instead of using the plain output arrays containing the panoptic-segmentation, we transform these arrays into a polygon representation of all things and the stuff instances. For the things, we could implement this by just taking the convex-hull enclosing the point cloud of the instance. However, with the stuff, we can not just follow this procedure, since point clouds representing the same stuff are assigned the same class, although they might be separated from each other. This makes perfect sense in the panoptic-segmentation, with stuff describing amorphous regions a human would assign the same meaning to (e.g. road, being something vehicles drive on), however, when taking the convex hull of the stuff, we would drastically extend their domains, with for example sidewalk polygons being closed across streets and similar unwanted side effects. To solve this problem we use the edge-filtering and the DBSCAN clustering algorithm [58] on the stuff in the images to separate stuff point clouds within a class, thus when closed by the convex-hull, we have two sidewalk instances with a road in between and not one sidewalk polygon being closed over the road. For outlier elimination and computational convenience, we also use this practice with thing instances in the image. The applied kernel is the simple 3×3 edge detection kernel and for the DBSCAN algorithm we set the distance to 2.5 and the number of neighbours to four[9]. A sketch of the process is drawn in fig. 18 and an example of an application of the algorithm is shown in fig. 17. The polygon representation also allows for an additional form of error correction, as we can threshold the surface area of a thing or stuff polygon to be accepted. By this, we can reduce the number of fragment defections that are spread over some panoptic-segmentations, predominantly in samples of DriveU. As threshold value we choose 400 px^2.

[9]We use the skikit-learn implementation of DBSCAN [11]


(a)

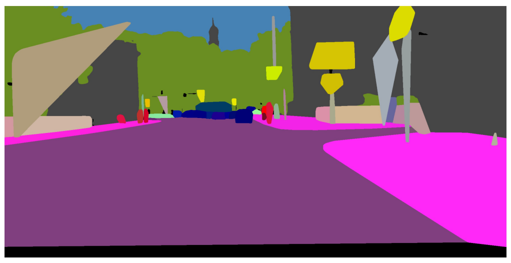

(b)

Fig. 17: The input to the algorithm is the panoptic-segmentation of fig. 16 (b). On the left-hand side, only the convex hull is formed, while on the right-hand side we applied the edge filtering and the DBSCAN algorithm as well as thresholding the minimum allowed area. In both cases we used the plain panoptic-segmentation for classes that are of no interest for the later analysis of pedestrian crossings (sec. 7), e.g. sky, vegetation and buildings. One can see the problems, if DBSCAN is not applied, with instances leaking over other instances and outliers, drastically enlarging instances when the convex hull is formed (a).

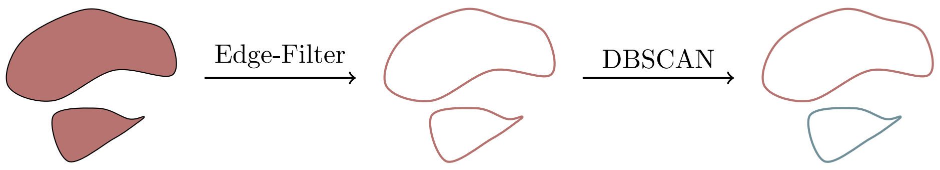


Fig. 18: The postprocessing pipeline for the panoptic-segmentation. The panoptic-segmentation is at first edge filtered, drastically reducing the points in a cloud, only leaving the enclosing points of an instance (may it be a thing or stuff). Then the DBSCAN algorithm is applied to all the different thing and stuff instances in the image to separate instances of the same class that have a margin in between them. After the DBSCAN algorithm, we extract the convex hull from the points, being our polygon for further analysis.

6.3 Results

The performance of the Panoptic-DeepLab pipeline with the HRNet+ backbone, on Cityscapes[10], reported by Cheng et al. in [57], is summarised in tab. 8. We can not evaluate the model on either the

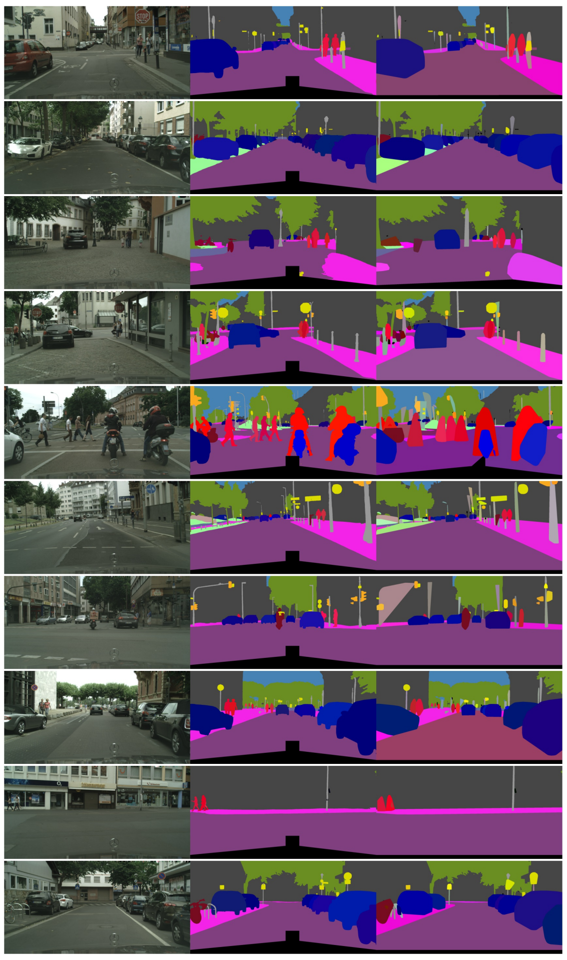
Tab. 8: The performance of the used Panoptic-DeepLab model on Cityscapes, reported in [57]. The explanation of the metrices can be found in [56].

PQ	SQ	RQ	AP	mIoU
63.4	81.5	76.7	29.9/29.6	80.9

DriveU nor the train-extra and test set of Cityscapes by comparing it to ground truth, as all of this datasets do not come equipped with a panoptic-segmentation, which is indeed the reason we deploy this model in the first place. However, what we can do is to look at a subset of the produced segmentations, and perform a qualitative evaluation of the results. Such a sample is shown in fig. 19. We can see that

[10]They do give no further specification about the subset the model is tested on, thus we suspect it to be the whole dataset (except the test set, as it is not publicly accessible).

the produced instance segmentations are descent and fit the human estimation for the stuff and thing instances quite well. Errors in the form of small fragmented pixels are often corrected in the postprocessing step, with, however, the problem, that correct instances that are at a huge distances get excluded as well (e.g. sec. 7 (a), second row). Another error appears threw the convex hull operation that still sometimes drastically enlarges instances, especially poles of traffic lights and signs (see fig. 17 (b))
The segmentation produced for DriveU samples is rather good, considering that the only preprocessing step is normalisation, with the classification making the same errors as on Cityscapes samples, and some additional classification errors on very bright and dark domains (e.g. sec. 7 (b), fourth row).


(a) Samples drawn from the Cityscapes train-extra and test set.

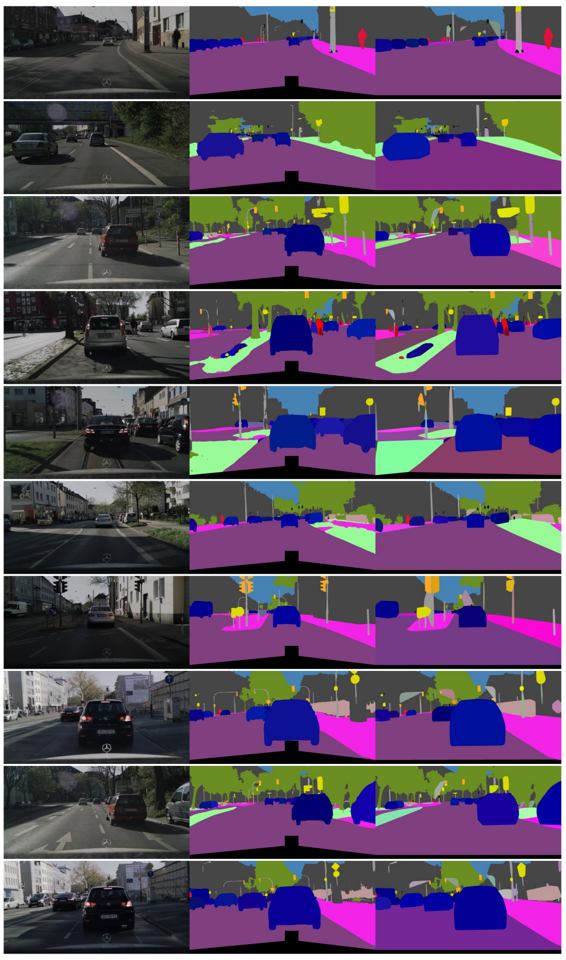

(b) Samples drawn from DriveU.

Fig. 19: Examples of the panoptic-segmentation produced by the Panoptic-DeepLab model and our postprocessing. On the left is the original image, in the middle the panoptic-segmentation in its raw representation and on the right our polygon representation, for the stuff and things of interest, mixed with the raw panoptic-segmentation, for the leftover classes.

7 Parsing the Datasets

The goal of this section is to construct a procedure that allows for the identification and evaluation of pedestrian crossings contained in the Cityscapes and DriveU datasets by utilising the models we developed and discussed in sec. 5 and sec. 6, as well as the ground truth and the depth of the samples. With an algorithm based on this procedure, we then produce a dataset of the identified pedestrian crossings, containing the state of the respective traffic lights and information about the presence of pedestrians on the street, together with the results leading to that identification and the pedestrian localisation.

7.1 Pedestrian Crossing Identification

Having the pedestrian crossings managed by a traffic light (PC) and being observed from the viewpoint of a vehicle, we are confronted with numerous configurations of which only a subset can be identified and analysed with the means of the data and the tools accessible to us. However, one of the most common configurations[11] of a PC can be selected rather accurate with our tools and hence is chosen as the object of our analysis. This PC is built from two vehicle-traffic lights appearing in a conglomerate with the respective pedestrian-traffic lights, on the left- and on the right-hand side of the street the observing vehicle is moving on. The traffic lights are located roughly three to four meters above ground and can appear together with several more, above the street or attached to one of the original lights, with all of letter, however, having neglectable importance for the identification and analysis of the PC.

Our identification of a pedestrian crossing will be limited solely to information (state, type, relevance, position, measures, depth) related to the traffic lights in the image. This makes the selection rather easy, but also vulnerable to systematic errors as we will see later. However, a more advanced study also relying on contextual and semantic information (of for example the road, sidewalks and poles) is beyond the scope of this thesis.

Even though we limited ourselves to a rather simple configuration we have to make a distinction that is not related to the PC itself but to its observability in the (image) data and leaves us with two different scenarios. In the first case, the vehicle (and therefore also the camera) is in a distance of the PC and the image contains the full setup described above. Whereas in the second scenario the vehicle is quite close to the PC and only one of the two traffic lights is visible in the image. Now in the first case, the state of the traffic lights on both sides should be the same while their type can differ (one of the traffic lights could, for example, have a circular-shaped light bulb while the other one has an arrow pictogram on it). As for the first case we do not demand that the traffic lights are relevant as the pair can be at a further distance and therefore not affecting the immediate path of the vehicle. Nonetheless, the relevance of both traffic lights should match (we loosen this condition for Cityscapes, due to the uncertainty in

[11]This conception is solely based on the experience of the author being a member of traffic and looking at the used datasets.

the relevance feature). In contrast to that, the single traffic light is expected to be quite close and hence should be relevant as it is immediately important for the planned path of the vehicle. For the first and also for the second case, the traffic lights should be located in the upper right (first traffic light) and upper left (second traffic light) domains of the image while also being below a certain distance to the car. With the distance threshold, we make sure that the pedestrian crossing is still visible and can be analysed with our means. The problem with the increasing distance of PCs is the decreasing accuracy in depth-estimation and classification (for the traffic lights in Cityscapes and the instance segmentation in DriveU), while also the likelihood of occlusion by vehicles and other obstacles increases. A list of criteria, as well as a scheme of the two scenarios, is shown in tab. 9.

One notices that we excluded a rather decisive criterion from the selection process being the presence of a pedestrian traffic light next to the vehicle traffic light. Hence, we no longer can distinguish between a PC and a crossroad where the traffic lights only manage vehicle traffic and no pedestrian crossing is present. The reason for this is quite simple and related to the labelling policies of the datasets. In DriveU the vehicle traffic lights are labelled with a quite high standard and almost none is left out from the labelling process, while with the pedestrian traffic lights the situation is different. They are often not annotated and labelled, especially if they appear sideways and in a conglomerate with a vehicle traffic light (see for example the pedestrian crossing in fig. 3). In Cityscapes the situation is different but no less problematic. As we already mentioned in sec. 4.3 traffic lights that are mounted on the same pole are often labelled as one instance and we, therefore, can not distinguish between vehicle and pedestrian traffic lights (see for example the traffic lights on the right- and left-hand side in fig. 4). Hence both datasets do not provide a level of accuracy and/or completeness in their data to include this feature into the selection.

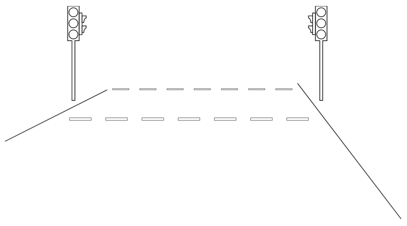
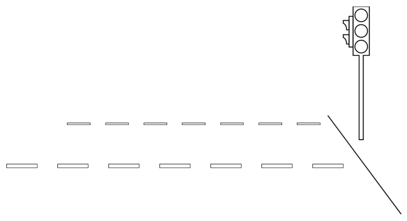
From the abstract conditions we previously formulated, we now want to develop an exact procedure for both datasets, that for a given sample (traffic lights in an image, with their class, i.e. type, state and relevance, their position, measures and their depth), decides if a pedestrian crossing is present and returns the traffic lights managing it.

7.1.1 Acquisition of the traffic light data

First, we need to obtain the data based on which we want to make our evaluation. Thus for a given sample (i.e. image and ground truth) of Cityscapes or DriveU, we acquire the following measures for every traffic light recognised.

Depth: As in sec. 5.2 we only assign a depth scalar to a traffic light. This scalar is obtain by calculating the median of the disparity values enclosed either by the bounding box (DriveU) or panoptic-segmentation polygon (Cityscapes) of the respective traffic light and then use the formula in eq. 5.16 to get the depth.

Position and Measures: By position and measures we mean the pixel position of the centre of the bounding box rectangle, as well as the width and height of the bounding box. With DriveU we can just use the ground truth while for Cityscapes we calculate the bounding box as the minimum rectangle enclosing the traffic light polygon of the panoptic-segmentation.

	Scenario 1	**Scenario 2**
		
Description:	The pedestrian crossing is at a distance and both traffic lights are visible in the image.	The pedestrian crossing is close to the car (and the camera), thus only one traffic light is visible in the image.
First traffic light (FTL)	- The FTL is below a certain distance to the car (depth threshold). - The FTL is a vehicle traffic light. - The FTL appears on the right-hand side of the image centre. - The FTL appears above the centre of the image.	- The FTL is below a certain distance to the car (depth threshold). - The FTL is a relevant vehicle traffic light. - It appears on the right-hand side of the image centre (we exclude single traffic lights on the left to reduce the error rate). - The FTL appears above the centre of the image. - It is the closest traffic light that satisfies the named conditions.
Second traffic light (STL)	- The STL is at roughly the same depth as the FTL (depth abbreviation threshold). - It is a vehicle traffic light in the same state as the FTL. - The STL appears on the left-hand side of the image centre. - The STL appears at roughly the same height as the FTL. - It forms, together with the FTL, the closest pair that satisfies the named conditions.	———

Tab. 9: Criteria for the identification of a traffic light controlled pedestrian crossings from positional, type, state and relevance information of the traffic lights.

Class: For DriveU the ground truth provides us with information about the state, type and relevance of the traffic light. For Cityscapes the situation is different, as this dataset does not come with class labels for the traffic lights. Hence we need to deploy our model developed in sec. 5.
We crop the image of the traffic light according to the minimum enclosing bounding box, resize and normalise it and pass it together with the normalised[12] position, measure and depth information into our traffic light classifier, which then estimates the type, state and relevance. Since we can not rely on perfect classification we discard traffic lights, that our classifier is not perfectly certain about. Namely the ones that fall below a confidence level of 0.96 for the state subclass and 0.9 for the type class. This high levels might seem rigorous but studies have shown that neural networks, especially CNNs are not very well calibrated and often overconfident with their prediction [52], an effect that is certainly not mitigated but rather worsened when transferring such models between datasets.

7.1.2 Setup

The (preselected) traffic lights in a sample are parsed in ascending order regarding their depth feature. If the selected traffic lights exceed the overall depth threshold:

$$d_{\text{max}}^{\text{DriveU}} = 100 \text{ m}, \qquad d_{\text{max}}^{\text{Cityscapes}} = 150 \text{ m}, \tag{7.1}$$

is not a vehicle traffic light, is below the image centre or its bounding box centre is not positioned in the left half of the image, it is discarded. We already named the reason for using the depth threshold, being the decrease in accuracy of our depth information and also the precision of the class labels we acquire by the panoptic-segmentation and traffic light classification algorithm. The decrease of accuracy happens due to the simple fact that with increasing distance the number of pixels associated with an object (traffic light, person, car, etc.) decreases and by this also our (and the algorithms) certainty about the object's properties and class. Especially the panoptic-segmentation needs to be quite accurate as our pedestrian localisation is based slolely on it. Therefore, we use a harsher depth threshold with DriveU since for DriveU the instance segmentation is done by the algorithm from sec. 6, which was only trained on Cityscapes, and we suspect a higher error rate when transferring the model between datasets.
Now if the traffic light does not fail the selection based on state and type we check if there exists a sibling traffic light fulfilling the conditions:

1. The sibling is roughly at the same depth as the first traffic light, with an abbreviation between the depth of the traffic lights allowed being

$$d_{\text{abbr}} = 8 \text{ m} \tag{7.2}$$

at max.

2. The sibling is at roughly the same height, meaning that the traffic light bounding boxes overlap along the y-direction.

[12] The normalisation for the crops, as well as the position, measure and depth are done with a mean and standard deviation calculated from all samples in the Cityscapes train set. For the image crops we normalise every colour channel individually.

3. The siblings bounding box centre is located in the left half of the image.

4. The sibling has the same state as the first traffic light and for DriveU also the same relevance label.

The reason that we do not use the relevance feature as a criterion, matching traffic lights in Cityscapes, is the weak prediction ability of our relevance classifier (sec. 5.3), in contrast to DriveU where we have access to hand labelled ground truth, that should not contain a significant amount of errors.
If a traffic light pair is found the parsing of the sample is finished and the sample gets assigned a pedestrian crossing managed by the found pair of traffic lights. If no sibling is found in all traffic lights in the sample the selected traffic light is checked for the possibility of being a single traffic light. This means, for both datasets, the traffic light must be below the threshold:

$$d_{\text{single}} = 30 \text{ m} \tag{7.3}$$

and be relevant. If it satisfies the conditions the according sample gets assigned a pedestrian crossing managed by the single traffic light. If it does not the traffic light gets discarded.
For the case of all traffic lights of a sample being parsed while no pedestrian crossing is found the sample gets an empty label. Using this procedure we obtain the closest PC in the image in the form of two or one traffic light and discard all the other possibly present ones. All the threshold values were intuitively initialised analysing, for example fig. 11. We then successively adjusted them, evaluating several results in every step.

7.2 Pedestrian Localisation

Now that we extracted prospective pedestrian crossings in the form of a single or sibling traffic lights from the datasets, we have to localise the pedestrians in the scene and develop criteria to decide if they are on the PC or not.
We do decide if a pedestrian is on the PC or not based on the two conditions:

1. The pedestrian should be inside a three-dimensional bounding box enclosing the pedestrian crossing.

2. The pedestrian should be on the road.

7.2.1 First Condition

To check the first condition we use the location (depth and x- and y-pixel-positions) of the traffic light(s) to make up this PC cuboid. However, due to the two scenarios in sec. 7.1 and only limited information in the second case, we have to make a distinction.
If we have access to both traffic lights of the PC we can create the boarders in depth, x- and y-pixel-position based on the two traffic lights. First we check if the pedestrian (i.e. its bounding box centre[13]) is inside a two dimensional area spanning from $(x_0, \min\{y_0, y_1\})$ to (x_1, b). In this context x_0 and x_1 are the rightmost or leftmost x-pixel-positions and y_0 and y_1 are the mean y-pixel-positions of the left or right traffic light respectively. b denotes the y-coordinate of the bottom of the image, in our case 1024.

[13]Bounding boxes are again created by enclosing the samples panoptic-segmentation with the smallest rectangle possible.

Second, we demand that the pedestrian is at roughly the same depth as the two traffic lights. We check the depth by using the depth scalars of the traffic lights, calculated in the previous section and create an x-dependent depth margin. The depth margin is defined utilising the linear function:

$$d(x) = a \cdot x + b, \tag{7.4}$$

with:

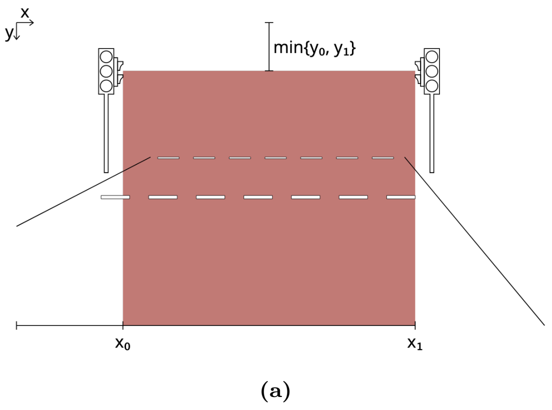
$$a = \frac{d_1 - d_0}{x_1 - x_0}, \tag{7.5}$$

$$b = d_0 - a \cdot x_0, \tag{7.6}$$

and the depth margin parameter d_m. If the pedestrian, with its mean bounding box position (x_p, y_p) and its depth d_p, calculated by taking the median over the values enclosed by the the pedestrians polygon in the depth map, satisfies:

$$d(x_p) - d_m \leq d_p \leq d(x_p) + d_m, \tag{7.7}$$

we assume that the pedestrian is at the traffic lights depth. We visualised the approach in fig. 20.
For the second scenario we are missing the information about a second traffic light, hence we must make



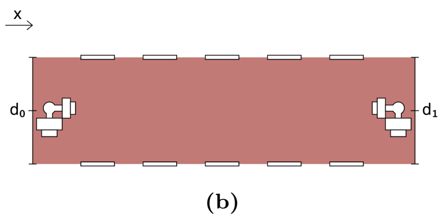


Fig. 20: Pedestrian localisation with both traffic lights being observed. (a) shows the valid area for pedestrians to be considered on the crosswalk regarding the x- and y-image-dimensions. The y_0, y_1 are the centre positions of the traffic light bounding boxes in the y-direction, x_0 and x_1 are the rightmost or leftmost coordinate of the traffic lights respectively. (b) visualises the depth margins for the pedestrians to be considered on the crosswalk. d_0 and d_1 are the depths of the traffic lights.

some assumptions and loosen our criteria. With only one traffic light present and suspecting another one beyond the left margin of the image we demand that the pedestrian (i.e. its bounding box centre) is located inside the area spanned by $(0, y_1)$ and (x_1, b), with y_1 being the mean y-pixel-position and the x_1 leftmost x-pixel-position of the bounding box enclosing the visible traffic light. Again b is the y-coordinate of the bottom image boarder, in our case 1024.
Now, we also again want the pedestrian to be at roughly the same depth as the recognised traffic light, however, missing the traffic light on the left, the creation of a function like eq. 7.4 is not possible. What

we do instead, is defining a new hyperparameter a_m called margin growth and the functions:

$$d_+(x) = a_m \cdot x + b_+, \tag{7.8}$$

$$d_-(x) = -a_m \cdot x + b_- \tag{7.9}$$

with:

$$b_+ = d_1 - a_m \cdot x_1, \tag{7.10}$$

$$b_- = d_1 + a_m \cdot x_1. \tag{7.11}$$

If the pedestrian satisfies:

$$d_-(x_p) - d_m \leq d_p \leq d_+(x_p) + d_m \tag{7.12}$$

we assume that the pedestrian is at the traffic lights depth. The depth trapezoid margine is used, due to the traffic light sibling possibly being shifted towards or away from the camera compared to its observed counterpart. This shifting might be caused by the tilting of the road and the position of the car so we use the margin growth to compansate for that. We visualised the practice in fig. 21.

Now if a pedestrian meets the properties for the respective scenario in the sample, described above, we

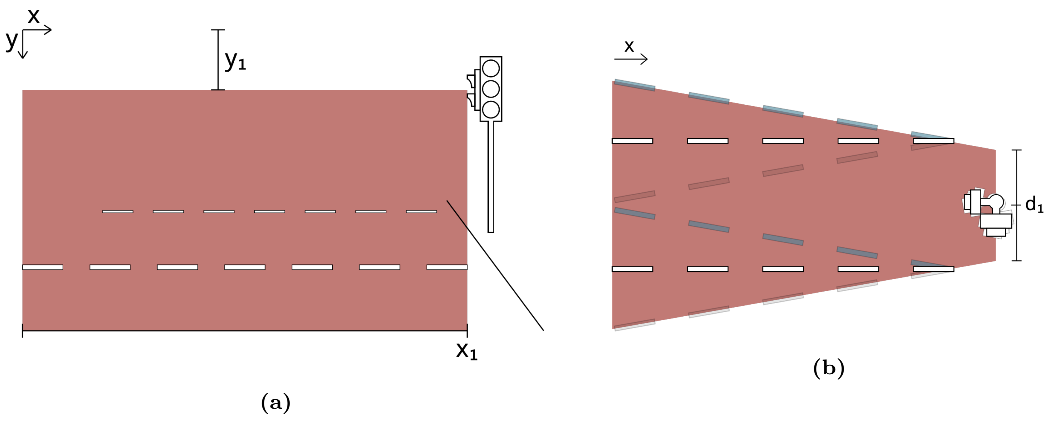


Fig. 21: Pedestrian localisation with one traffic lights being observed. (a) shows the valid area for pedestrians to be considered on the crosswalk regarding the x- and y-image-dimensions. The y_1 is the centre positions of the traffic light bounding boxes in the y-direction, while x_1 is the leftmost coordinate of the observed traffic light. (b) visualises the depth margins for the pedestrians to be considered on the crosswalk, with d_1 being the depth of the TL. We marked the possibly tilted alternatives of the PC, which, with our resources, are not distinguishable from a pedestrian crossing with a uniform depth.

consider condition one fulfilled.

7.2.2 Second Condition

To check the second condition we use our panoptic-segmentation, in particular our polygon representation of pedestrians, vehicles and ground domains of DriveU and Cityscapes. Now if a pedestrian fulfils the first condition we try to determine if his feet are on the road or not. This is done by first sorting the polygon vertices descending in their y-magnitude and then select at least the n_{min} first points. If those

points are colinear, we successively add further points, one at a time, until we obtain a list that is not colinear anymore.
We then form the convex hull of the selected points, assuming this technique provides us with a polygon enclosing the feet of the respective pedestrian.
Finally, we check if the obtained polygon intersects with a road polygon, and further if it intersects with a polygon of a list of exclusive classes. If the feet polygon does indeed intersect with the road and with no class from the exclusive list, we assume the pedestrian to be on the road, and hence the first condition fullfilled, otherwise not.

7.2.3 Setup

Checking the two conditions for valid pedestrian instances in a sample where a PC is recognised, allows us to obtain an estimated count of pedestrians as well as their location on the PC.
Valid means that the pedestrian polygon has a surface area. Regarding Cityscapes, we demand that this area is greater 0, while for DriveU we threshold the minimum area with 1000 px^2. The high threshold for DriveU is applied to reduce misclassifications, assumed to be more frequent in DriveU, due to the transfer of the panoptic-segmentation model.
For both the DriveU and the Cityscapes dataset we use:

$$d_m = 4 \text{ m}, \qquad a_m = \frac{3}{1024} \text{ m}, \tag{7.13}$$

thus the depth margin, in the most extreme case, grows to 14 meters. The exclusive list, for both datasets, contains the classes: sidewalk, parking, terrain, ground, car, truck, bus, motorcycle, caravan, trailer, bicycle, traffic sign, fence and pole.
All the above parameters were again selected by a successive change of them observing the obtained results in every step. After the selection we parse the DriveU and Cityscapes datasets identifying pedestrian crossings and localising pedestrians. From the obtained and used information, we create datasets for DriveU, Cityscapes train and the Cityscapes test and extra sets.

8 Results

In this chapter, we present and discuss the results obtain using the procedure developed in sec. 7 on the DriveU and Cityscapes datasets and perform a false-positive error correction.
For DriveU we parsed 40.978 samples and our algorithm detected 17.953 pedestrian crossings. In the case of Cityscapes, we parsed 3.475 samples in the trainset and found 423 pedestrian crossings and 21.522 samples in the test and train-extra set with 1.822 pedestrian crossing detections. Looking at the plain results, i.e. the number of detections, one notices the discrepancy between the detection rate for DriveU (about 0.44) compared to Cityscapes (about 0.09). This difference might be caused by two reasons. The first reason is the creation of DriveU as a dataset containing traffic light controlled crossings, with the camera only recording when approaching such a system [24]. Hence, if compared to Cityscapes, which does not perform a selection of any specific urban- and suburban-scenes, the frequency of PCs in DriveU should outnumber the one in Cityscapes. The second reason is the pedestrian crossing identification being solely reliant on the information about the position, state, type and relevance of the traffic lights in an image. For DriveU we have a ground truth, while for Cityscapes the types, state and relevance information are obtained using our TL classifier producing a higher rate of missclassifications (tab. 17), and therefore, probably also a decrease in detection rate.
The plain probabilities are listed in tab. 10. The variable P indicates the presence of pedestrians on the crossing and the variable S the state of the managing traffic light(s).

Dataset	P(P=yes, S=red)	P(P=yes, S=red-yellow)	P(P=yes, S=yellow)	P(P=yes, S=green)
DriveU	0.014872	0.000167	0.000446	0.003732
Cityscapes train	0.101655	0.000000	0.002364	0.004728
Cityscapes train-extra, test	0.029089	0.000000	0.000549	0.004391
all	0.017972	0.000149	0.000495	0.003812

Dataset	P(P=no, S=red)	P(P=no, S=red-yellow)	P(P=no, S=yellow)	P(P=no, S=green)
DriveU	0.235448	0.019551	0.034089	0.691695
Cityscapes train	0.411348	0.009456	0.014184	0.456265
Cityscapes train-extra, test	0.343578	0.010428	0.022503	0.589462
all	0.248886	0.018517	0.032627	0.677542

Tab. 10: The uncorrected joint probabilities, we obtained from parsing the datasets. The first random variable states the presence of a pedestrian on the crossing and the second one the state of the traffic light.

8.1 Error Correction

To correct for errors and get an estimate for the performance of our approach we conduct an error correction. For every categories and the three sets, DriveU, Cityscapes train and Cityscapes train-extra and test, we analyse up to 100 samples per configuration (i.e. pedestrian on the street or not, state of the traffic light) to get an estimate for false-positive errors of our algorithm and correct the obtained probabilities accordingly. We also examine the frequency of the error type, forming three categories:

A segmentation error means that the primary error is cause by the segmentation of the sample, hence, either the pedestrians in the image are not annotated accurate enough, or areas are classified as road, although they are not (fig. 22a). Also if there are miss-detection of traffic lights, that classify areas as traffic light with none present in the image, fall into that category.

A traffic light classification error is an error that is caused by a missclassification of the traffic lights in the image. For example if our class prediction for a traffic light says green although the image shows clearly a red traffic light. However, the category also contains wrong identifications of pedestrian crossings, meaning that a crossing is detected although none present (fig. 22b). For those errors we decide if the false detection could have been avoided, with a correct classification, and if so, the error falls in the traffic light classification error category.

The localisation and identification error is an error which is caused not by the provided information, but by our choices of how to identify a PC and localise pedestrians. Hence this category contains errors where for example traffic lights above the street, instead of left and right of the street are selected (fig. 22c) or traffic lights that are not related to each other form, according to our identification scheme, a pedestrian crossing (fig. 22d). Also assigned to this category is the miss localisation of pedestrians on the PC, where there is no error in the segmentation, but the error is in the localisation procedure of the pedestrians (fig. 22e).

The categories are mutually exclusive, hence if numerous errors are present, we try to identify the dominant error, and assign the error according to our perception of the primary error. The counts for the errors are summarised in tab. 17 under sec. B in the appendix. In tab. 11 we summarised the fraction

Dataset	Segmentation Error	TL Classification Error	Loc. and Ident. Error	Segmentation	TL Labels
DriveU	67 %	6 %	27 %	Panoptic-DeepLab	ground truth
Citysapes train	0 %	41 %	59 %	ground truth	TL classifier
Cityscapes train-extra, test	10 %	35 %	55 %	Panoptic-DeepLab	TL classifier

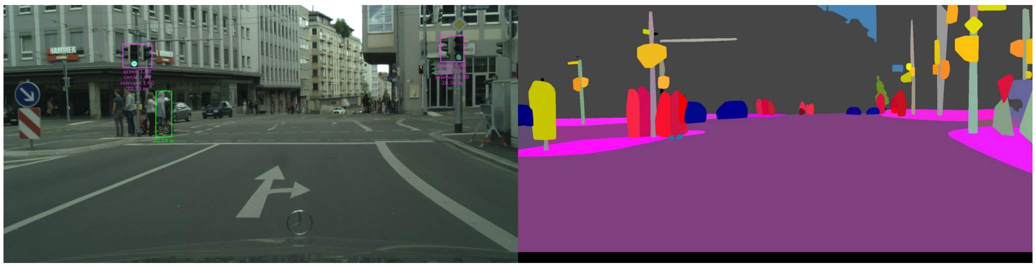
Tab. 11: The composition of the errors made and the sources of the used data.

of errors made in each category and also added the data source of the segmentation and the traffic light labels.

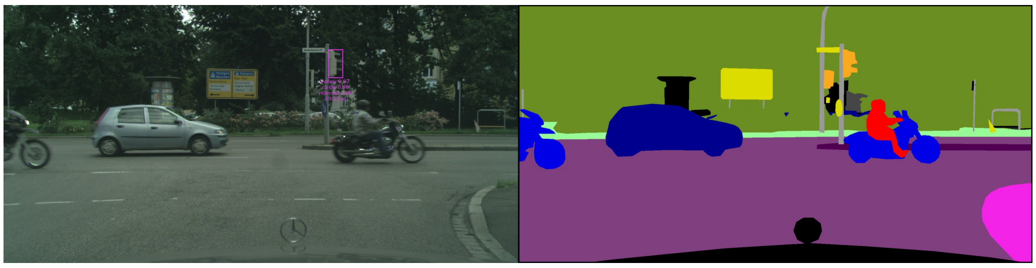
We notice that the segmentation error dominates for DriveU, being what we already expected with the transfer of the Panoptic-DeepLab model from Cityscapes to DriveU samples. For Cityscapes train, train-extra and test the fraction of localisation and identification error dominates, followed by the TL classification error, caused by the TL classifier, we trained on DriveU and deployed on Cityscapes. On DriveU we get an estimate of 18.26 % of the total false-positive error rate, on Cityscapes train it is 17.97 % and on Cityscapes train-extra and test 17.08 % leading to a total error rate of 17.87 %

Now, with our estimates of the false-positive error rates we can correct the probabilities in tab. 10, reducing the number of samples in a category by the expected amount of false-positives in that category ($r_{categ.}$), obtained from tab. 17. Hence the new number of samples in a category is then:

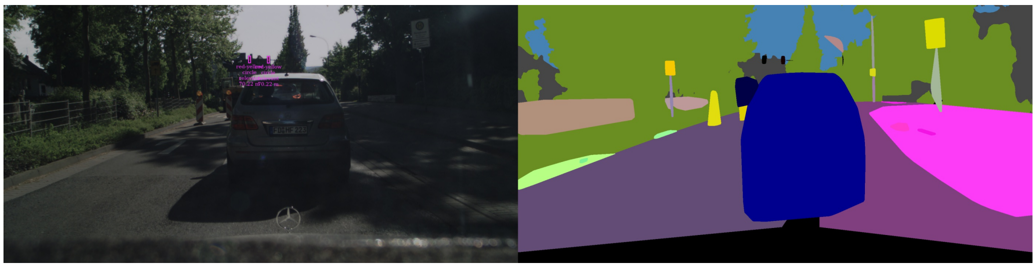
$$n'_{categ} = (1 - r_{categ.})\, n_{categ}. \tag{8.1}$$



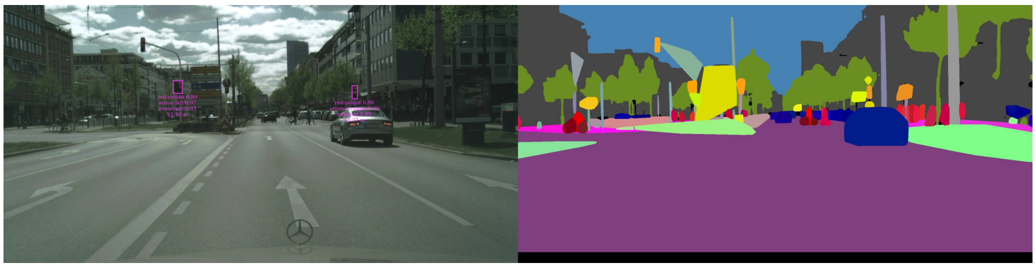
(a) A segmentation error, with the road extending onto the sidewalk, leading to a pedestrian wrongfully located on the PC. The blue scatter in the segmentation image indicates the points used to create the feet-polygon of the detected pedestrian.



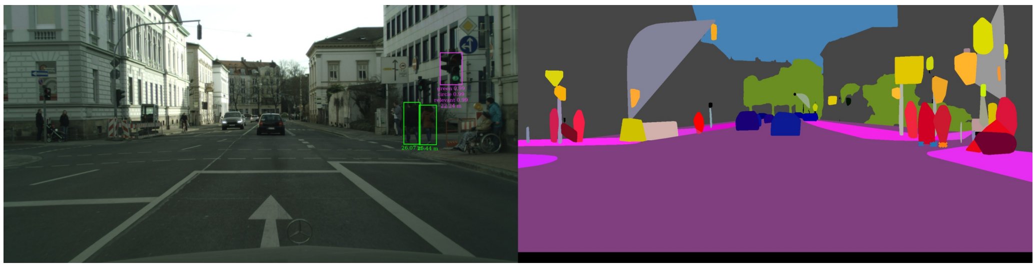
(b) A traffic light classification error, with a missclassification of a traffic light leading to a false detection of a PC.



(c) A localisation and identification error. The traffic lights above the street are chosen and identified as PC.



(d) A localisation and identification error. Two unrelated traffic light are falsely joined to a pedestrian crossing.



(e) A localisation and identification error. The pedestrians are on the street and the PC is identified correct. However, the pedestrians are not crossing the selected PC, but a road perpendicular.

Fig. 22: Samples for the different error categories.

With the new counts in every category we recalculate the probabilities, which are summarised in tab. 12. Finally those corrected results can be used to estimate the probability of a critical situation, i.e. the

Dataset	P(P=yes, S=red)	P(P=yes, S=red-yellow)	P(P=yes, S=yellow)	P(P=yes, S=green)
DriveU	0.014472	0.000058	0.000058	0.000408
Cityscapes train	0.119834	0.000000	0.002923	0.005846
Cityscapes train-extra, test	0.032743	0.000000	0.000000	0.001284
all	0.017857	0.000052	0.000105	0.000577

Dataset	P(P=no, S=red)	P(P=no, S=red-yellow)	P(P=no, S=yellow)	P(P=no, S=green)
DriveU	0.224189	0.018002	0.033529	0.709282
Cityscapes train	0.371252	0.011691	0.014614	0.473841
Cityscapes train-extra, test	0.301429	0.007704	0.022471	0.634369
all	0.233143	0.017047	0.032286	0.698933

Tab. 12: The corrected joint probabilities, we obtained from parsing the datasets.

traffic light is green and there are pedestrians on the street:

$$\mathrm{P(P{=}yes \mid S{=}green)} = \frac{\mathrm{P(P{=}yes,\ S{=}green)}}{\mathrm{P(P{=}yes,\ S{=}green) + P(P{=}no,\ S{=}green)}}. \tag{8.2}$$

The results are listed in tab. 13. We also attached the samples for the P=yes and S=green state and depicted them in fig. 25 under sec. C.

	P(P=yes | S=green)
DriveU	0.000575
Cityscapes train	0.012187
Cityscapes train-extra, test	0.002020
all	0.000825

Tab. 13: The conditional probability of a pedestrian on the street given the traffic light is green. The first random variable states the presence of a pedestrian on the crossing and the second one the state of the traffic light.

9 Discussion and Conclusion

In this work, we examined the scenario of a pedestrian crossing managed by a traffic light, based on data from urban- and suburban-scenes. To identify those PCs in the images of the DriveU and Cityscapes dataset and extract information about the state of the managing TLs as well as the presence of pedestrians, we, on the one hand, used the ground truth of the named datasets, and on the other hand, utilised machine learning models to obtain the needed information from image data.

The traffic light classification model was trained on DriveUC train and then tested on DriveUC test, achieving an overall sample weighted f-score of 0.7423 and 0.9397 in the state category. For the panoptic-segmentation we modified the pre-trained Panoptic-DeepLab pipeline to fit our needs and examined some result of the model and the engineered post-processing fig. 19 on DriveU and Cityscapes.

Finally, we formulated a set of rules incorporated in a procedure (i.e. an algorithm), to count the number of pedestrian crossings in the dataset and extract the state information of the managing traffic light as well as localising pedestrians on the PCs (sec. 7).

The overall false-positive error rate of the pedestrian crossing identification and the pedestrian localisation (also including errors in the underlying data) is about 18 %, and after correction, we report the probability for a critical situation (i.e. given a green traffic light, what is the likelihood for the presence of pedestrians on the PC) with 0.0825 %.

The accuracy of this probability is, however, hard to evaluate, as we only have an estimate for the false-positive errors and we do not know the false-negative error rates. If our algorithm had a uniform false-negative error rate for all setups there would not be a problem, as for every setup the same ratio of samples would be left out. Yet, such a scenario is highly unlikely, with the false-positive errors distributed rather unequally (tab. 17) and the amount of information needed to decide for a setup, differing from setup to setup. Thus, to classify an empty PC with the traffic light in a green state we only need a correct traffic light localisation and prediction as well as rather accurate depth information. For a setup with a pedestrian involved and a traffic light state that has a lower classification accuracy, for example, red-yellow, the information exceeds the previous case, as we now also need a correct pedestrian localisation. Hence, with the lower classification accuracy and the intuitive assumption that with increasing information needed to identify a setup, the error rate grows, there should be a discrepancy in error rates for the different setups, not only for the false-positives, as we observed, but also for the false negatives.

An examining of false-negatives, by observing samples that our procedure discarded, would therefore be a task that could improve the value of the calculated probabilities, by allowing for false-negative error estimation. Besides that, we also thought about some other changes or improvements not directly regarding the results of our approach, but the approach itself.

First, one could improve the performance of the used networks, using more advanced alteration schemes and also retrain the models on either some samples of DriveU (for the Panoptic-DeepLab model) or Cityscapes (for the traffic light classifier) to facilitate the transfer.

However, instead of using the old traffic light classifier, we propose to classify the type, state and relevance with a model that operates on the full 2048×1024 images instead of the crops. Doing so would, on the one hand, make the creation of a separate dataset (DriveUC) obsolete, mitigating the error made due to the difference in the DriveUC and the Cityscapes traffic light crops, while on the other hand probably improving the relevance classification. The improvement in the relevance feature classification, we believe, is tied to the nature of the feature itself, with it being closely related to contextual information in the image, i.e. the orientation of the car and direction of the road, the placement of the traffic light and other traffic lights present in the image. A lot of this information gets lost when we use our approach, whereat with a network working on the full-sized images, it is not.

To improve the pedestrian localisation, one could give pedestrian instances in the Cityscapes train segmentation an additional label, based on the point where they stand (i.e. on the road/not on the road) and then retrain the pre-trained Panoptic-DeepLab pipeline also classifying this additional label. Doing so one could circumvent the very error sensitive[14] procedure used to check if the pedestrian is on the road or not, while also making the postprocessing obsolete. An approach to prevent errors like the one depicted in fig. 22e, with pedestrians walking on perpendicular roads, on could perform a pose estimation and discard all pedestrians that are headed in a different direction than across the PC.

Finally one could improve the selection process of pedestrian crossings by including criteria like the road orientation and main pole axis the traffic light are attached to. With the poles, one could exclude the error of selecting two traffic lights above the street, by looking for example at the main orientation of the poles in the close neighbourhood of the TL. If this orientation is close to vertical, we keep the traffic light, otherwise, we discard it. With the road orientation, one could probably mitigate errors, like in fig. 22b, by discarding samples where the road orientation is close to horizontal.

Now in the course of this work, have shown that the accessibility of data as well as the advancement of machine learning models, allows for the analysis of a rigid scenario in urban- and suburban-scenes and named some improvements that could be applied to our approach. Further work could, therefore, either improve the used approach or search for other critical scenarios in urban- and suburban-scene datasets.

[14]The main error in the segmentation category in tab. 17 is caused by wrongfully localising pedestrians on the road.

Glossary

ASPP	Atrous spacial pyramid pooling. 36, 37
bn	Batch normalisation. 22, 23
CNN	Convolutional neural network. 9, 10, 44
conv	Convolution. 22, 23
FNN	Feedforward neural network. 5, 6, 9, 10
i.i.d.	Independent and identically distributed. 5, 24
MLE	Maximum likelihood estimate. 8
NLL	Negative logarithmic likelihood. 8
PC	Pedestrian crossing managed by a traffic light system. 41, 42, 45, 47–51, 53, 54, 68
ReLU	Rectified linear unit. 6, 10, 22, 23, 30
sigmoid	The sigmoid function. 30
softmax	Normalized exponential function. 7, 22
TL	Traffic light. 47, 49, 50, 53, 54

References

[1] T. M. Mitchell, *Machine learning*, eng, Internat. ed., [Nachdr.] New York [u.a.]: McGraw-Hill, 2007, XVII, 414 S. ISBN: 978-0-07-115467-3.

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

[3] C. Bishop, *Pattern Recognition and Machine Learning*. Springer, Jan. 2006. [Online]. Available: `https://www.microsoft.com/en-us/research/publication/pattern-recognition-machine-learning/`.

[4] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" In *Proc. International Conference on Computer Vision (ICCV'09)*, IEEE, 2009.

[5] K. P. Murphy, *Machine learning, a probabilistic perspective*, eng. Cambridge, Mass. [u.a.]: MIT Press, 2012, XXIX, 1071 S. ISBN: 978-0-262-01802-9.

[6] L. Ardizzone, C. Lüth, J. Kruse, C. Rother, and U. Köthe, *Guided image generation with conditional invertible neural networks*, 2019. arXiv: `1907.02392 [cs.CV]`.

[7] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2014. arXiv: `1412.6980 [cs.LG]`.

[8] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, 2015. arXiv: `1502.03167 [cs.LG]`.

[9] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, *How does batch normalization help optimization?* 2018. arXiv: `1805.11604 [stat.ML]`.

[10] J. Kohler, H. Daneshmand, A. Lucchi, M. Zhou, K. Neymeyr, and T. Hofmann, *Exponential convergence rates for batch normalization: The power of length-direction decoupling in non-convex optimization*, 2018. arXiv: `1805.10694 [stat.ML]`.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[12] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *Tensorflow: Large-scale machine learning on heterogeneous distributed systems*, 2016. arXiv: `1603.04467 [cs.DC]`.

[13] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, *Pytorch: An imperative style, high-performance deep learning library*, 2019. arXiv: `1912.01703 [cs.LG]`.

[14] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, *Densely connected convolutional networks*, 2016. arXiv: `1608.06993 [cs.CV]`.

[15] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual.* Scotts Valley, CA: CreateSpace, 2009, ISBN: 1441412697.

[16] G. J. Brostow, J. Shotton, J. Fauqueur, and R. Cipolla, "Segmentation and recognition using structure from motion point clouds," in *ECCV (1)*, 2008, pp. 44–57.

[17] R. de Charette and F. Nashashibi, *Real time visual traffic lights recognition based on spot light detection and adaptive traffic lights templates*, 2009.

[18] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013. DOI: `10.1177/0278364913491297`. eprint: `https://doi.org/10.1177/0278364913491297`. [Online]. Available: `https://doi.org/10.1177/0278364913491297`.

[19] T. Scharwächter, M. Enzweiler, U. Franke, and S. Roth, "Efficient multi-cue scene segmentation," in *GCPR*, 2013.

[20] M. B. Jensen, M. P. Philipsen, A. Møgelmose, T. B. Moeslund, and M. M. Trivedi, "Vision for looking at traffic lights: Issues, survey, and perspectives," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 7, pp. 1800–1815, 2016. DOI: `10.1109/TITS.2015.2509509`.

[21] M. P. Philipsen, M. B. Jensen, A. Møgelmose, T. B. Moeslund, and M. M. Trivedi, "Traffic light detection: A learning algorithm and evaluations on challenging dataset," in *intelligent transportation systems (ITSC), 2015 IEEE 18th international conference on*, IEEE, 2015, pp. 2341–2345.

[22] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," *CoRR*, vol. abs/1604.01685, 2016. arXiv: `1604.01685`. [Online]. Available: `http://arxiv.org/abs/1604.01685`.

[23] K. Behrendt and L. Novak, "A deep learning approach to traffic lights: Detection, tracking, and classification," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, IEEE.

[24] A. Fregin, J. Müller, U. Krebel, and K. Diermayer, "The driveu traffic light dataset: Introduction and comparison with existing datasets," in *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, IEEE, 2018, pp. 3376–3383. DOI: `10.1109/ICRA.2018.8460737`. [Online]. Available: `https://doi.org/10.1109/ICRA.2018.8460737`.

[25] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, *Microsoft coco: Common objects in context*, 2014. arXiv: `1405.0312 [cs.CV]`.

[26] M. Weber, P. Wolf, and J. M. Zöllner, "Deeptlr: A single deep convolutional network for detection and classification of traffic lights," in *2016 IEEE Intelligent Vehicles Symposium (IV)*, 2016, pp. 342–348.

[27] G. Trehard, E. Pollard, B. Bradai, and F. Nashashibi, "Tracking both pose and status of a traffic light via an interacting multiple model filter," Jul. 2014.

[28] L. C. Possatti, R. Guidolini, V. B. Cardoso, R. F. Berriel, T. M. Paixão, C. Badue, A. F. De Souza, and T. Oliveira-Santos, "Traffic light recognition using deep learning and prior maps for autonomous cars," in *2019 International Joint Conference on Neural Networks (IJCNN)*, 2019, pp. 1–8.

[29] A. Gómez Hernandez, F. Ribeiro de Alencar, P. Prado, F. Osorio, and D. Wolf, "Traffic lights detection and state estimation using hidden markov models," Jun. 2014, pp. 750–755, ISBN: 978-1-4799-3638-0. DOI: `10.1109/IVS.2014.6856486`.

[30] Z. Cai, Y. Li, and M. Gu, "Real-time recognition system of traffic light in urban environment," in *2012 IEEE Symposium on Computational Intelligence for Security and Defence Applications*, 2012, pp. 1–6.

[31] C. Jang, C. Kim, D. Kim, M. Lee, and M. Sunwoo, "Multiple exposure images based traffic light recognition," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, 2014, pp. 1313–1318.

[32] D. Nienhüser, M. Drescher, and J. M. Zöllner, "Visual state estimation of traffic lights using hidden markov models," in *13th International IEEE Conference on Intelligent Transportation Systems*, 2010, pp. 1705–1710.

[33] C.-C. Chiang, M.-C. Ho, H.-S. Liao, A. Pratama, and W.-C. Syu, "Detecting and recognizing traffic lights by genetic approximate ellipse detection and spatial texture layouts," *International Journal of Innovative Computing, Information and Control*, vol. 7, pp. 6919–6934, Dec. 2011.

[34] R. de Charette and F. Nashashibi, "Real time visual traffic lights recognition based on spot light detection and adaptive traffic lights templates," in *2009 IEEE Intelligent Vehicles Symposium*, 2009, pp. 358–363.

[35] ——, "Traffic light recognition using image processing compared to learning processes," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009, pp. 333–338.

[36] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, *Imagenet large scale visual recognition challenge*, 2014. arXiv: `1409.0575 [cs.CV]`.

[37] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017, ISSN: 0001-0782. DOI: `10.1145/3065386`. [Online]. Available: `https://doi.org/10.1145/3065386`.

[38] J. Long, E. Shelhamer, and T. Darrell, *Fully convolutional networks for semantic segmentation*, 2014. arXiv: `1411.4038 [cs.CV]`.

[39] F. Lindner, U. Kressel, and S. Kaelberer, "Robust recognition of traffic signals," in *IEEE Intelligent Vehicles Symposium, 2004*, 2004, pp. 49–53.

[40] V. John, K. Yoneda, B. Qi, Z. Liu, and S. Mita, "Traffic light recognition in varying illumination using deep learning and saliency map," in *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, 2014, pp. 2286–2291.

[41] V. John, K. Yoneda, Z. Liu, and S. Mita, "Saliency map generation by the convolutional neural network for real-time traffic light detection using template matching," *IEEE Transactions on Computational Imaging*, vol. 1, no. 3, pp. 159–173, 2015.

[42] U. Franke, D. Pfeiffer, C. Rabe, C. Knoeppel, M. Enzweiler, F. Stein, and R. G. Herrtwich, "Making bertha see," in *2013 IEEE International Conference on Computer Vision Workshops*, 2013, pp. 214–221.

[43] *Densenet citations*, Reviewed 07/11/2020. [Online]. Available: `https://ui.adsabs.harvard.edu/abs/2016arXiv160806993H/citations`.

[44] *Imagenet ranking*, Reviewed 07/11/2020. [Online]. Available: `https://paperswithcode.com/sota/image-classification-on-imagenet`.

[45] *Cifar-10 ranking*, Reviewed 07/11/2020. [Online]. Available: `https://paperswithcode.com/sota/image-classification-on-cifar-10`.

[46] *Cifar-100 ranking*, Reviewed 07/11/2020. [Online]. Available: `https://paperswithcode.com/sota/image-classification-on-cifar-100`.

[47] Y. LeCun, L. Bottou, G. Orr, and K. Muller, "Efficient backprop," in *Neural Networks: Tricks of the trade*, G. Orr and M. K., Eds., Springer, 1998.

[48] J. Snoek, H. Larochelle, and R. P. Adams, *Practical bayesian optimization of machine learning algorithms*, 2012. arXiv: `1206.2944 [stat.ML]`.

[49] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. 10, pp. 281–305, 2012. [Online]. Available: `http://jmlr.org/papers/v13/bergstra12a.html`.

[50] R. Eldan and O. Shamir, *The power of depth for feedforward neural networks*, 2015. arXiv: `1512.03965 [cs.LG]`.

[51] G. Montúfar, R. Pascanu, K. Cho, and Y. Bengio, *On the number of linear regions of deep neural networks*, 2014. arXiv: `1402.1869 [stat.ML]`.

[52] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, *On calibration of modern neural networks*, 2017. arXiv: `1706.04599 [cs.LG]`.

[53] B. Cheng, M. D. Collins, Y. Zhu, T. Liu, T. S. Huang, H. Adam, and L.-C. Chen, *Panoptic-deeplab: A simple, strong, and fast baseline for bottom-up panoptic segmentation*, 2019. arXiv: `1911.10194 [cs.CV]`.

[54] J. Wang, K. Sun, T. Cheng, B. Jiang, C. Deng, Y. Zhao, D. Liu, Y. Mu, M. Tan, X. Wang, W. Liu, and B. Xiao, *Deep high-resolution representation learning for visual recognition*, 2019. arXiv: `1908.07919 [cs.CV]`.

[55] *Official ranking on the panoptic-segmenation task on cityscapes*, Reviewed 07/05/2020. [Online]. Available: `https://www.cityscapes-dataset.com/benchmarks/#panoptic-scene-labeling-task`.

[56] A. Kirillov, K. He, R. Girshick, C. Rother, and P. Dollár, *Panoptic segmentation*, 2018. arXiv: `1801.00868 [cs.CV]`.

[57] *The github reposatory containing the panoptic-deeplab model*, Reviewed 09/07/2020. [Online]. Available: `https://github.com/bowenc0221/panoptic-deeplab`.

[58] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," AAAI Press, 1996, pp. 226–231.

Appendix

A Algorithms

Algorithm 1 *PathSearch.* The algorithm is configured for a model that should be minimised on the criterion. However, this can be changed by initialising vs_{best} with "$-\infty$" and changing the relation in line 9. to "$>$".

```
Global:
    $\mathcal{M}$, $\mathcal{C}$, $\mathcal{O}$                          // model, criterion, optimizer
    $\mathcal{D}_{train}$, $\mathcal{D}_{val}$                           // training and validation data
Input:
    $p$, $\mathcal{P}$                                                   // hyperparameter, hyperparameter alterations
    $p_{best} \leftarrow$ none                                           // best hyperparameter, initilised with none
    $vs_{best} \leftarrow +\infty$                                       // best validation score, initialised with infinity
Output:
    $p_{best}, vs_{best}$

 1: function PathSearch($p, \mathcal{P}, p_{best}, vs_{best}$)
 2:     $\mathcal{M}_p \leftarrow$ apply($\mathcal{M}, p$)                   // initilise model,
 3:     $\mathcal{C}_p \leftarrow$ apply($\mathcal{C}, p$)                   // criterion,
 4:     $\mathcal{O}_p \leftarrow$ apply($\mathcal{O}, p$)                   // and optimizer with their hyperparameters
 5:     $\mathcal{M}'_p \leftarrow$ train($\mathcal{M}_p$, $\mathcal{C}_p$, $\mathcal{O}_p$, $\mathcal{D}_{train}$)   // train model on the given configurations
 6:     $vs \leftarrow$ eval($\mathcal{M}'_p$, $\mathcal{C}_p$, $\mathcal{D}_{val}$)   // evaluate trained model on the validation data
 7:     if $vs < vs_{best}$ then                                         // compare validation scores
 8:         $vs_{best} \leftarrow vs$                                    // save the new best score
 9:         $p_{best} \leftarrow p$                                      // save the new best hyperparameter
10:         shuffle($\mathcal{P}$)
11:         for alteration in $\mathcal{P}$ do                           // random alteration according to $\mathcal{P}$
12:             $p' \leftarrow$ apply($p$, alteration)
13:             $p_{best}, vs_{best} \leftarrow$ PathSearch($p'$, $\mathcal{P}$, $p_{best}$, $vs_{best}$)
14:         end for
15:     end if
16:     return $p_{best}, vs_{best}$                                     // return best hyperparameters and validation score
17: end function
```

Algorithm 2 *earlyStopping.*

Input:

```
$E$, $V_{scores}$        // The epoch data and the validation scores data of the training
$I$                      // a tuple that specifies the regression interval
$s_{max}$                // maximum allowed slope
$f_{size}$               // size of the mean filter
```

Output:

```
bool
```

```
1: function earlyStopping($E$, $V_{scores}$, $I$, $s_{max}$, $f_{size}$)
2:     $V_{scores} \leftarrow$ meanFilter($V_{scores}$, $f_{size}$)        // mean filter the score
3:     $a, b \leftarrow$ linReg($E$, $V_{scores}$, $I$)        // $a$ is the slope and $b$ the bias
4:     if $a > s_{max}$ then
5:         return true
6:     else
7:         return false
8:     end if
9: end function
```

Algorithm 3 *Adam.* The algorithm is taken from [7]. g_t^2 is the elementwise square $g_t \odot g_t$ and $\beta_{1/2}^t$ denote $\beta_{1/2}$ to the power of t respectively.

```
Input:
    $\alpha$                                   // Stepsize
    $\beta_1, \beta_2 \in [0, 1)$              // Exponential decay rates for the moment estimates
    $\epsilon$                                 // Parameter for numeric stability
    $\mathcal{L}(\boldsymbol{\theta})$         // Stochastic objective function with parameter $\boldsymbol{\theta}$
    $\boldsymbol{\theta}_0$                    // Initial parameter vector
Output:
    $\boldsymbol{\theta}_t$                    // The resulting parameters

 1: function ADAM($\mathcal{L}, \beta_1, \beta_2, \epsilon, \boldsymbol{\theta}_0$)
 2:     $m_0 \leftarrow 0$                     // Initialise first moment vector
 3:     $v_0 \leftarrow 0$                     // Initialise second moment vector
 4:     $t \leftarrow 0$                       // Initialise time step
 5:     while $\boldsymbol{\theta}_0$ not converged do
 6:         $t \leftarrow t + 1$
 7:         $g_t \leftarrow \nabla_{\boldsymbol{\theta}} \mathcal{L}_t(\boldsymbol{\theta}_{t-1})$     // Get gradients w.r.t stochastic objective at timestep $t$
 8:         $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$     // Update biased first moment estimate
 9:         $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$   // Update bias second raw moment estimate
10:         $\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$     // Compute bias-corrected first moment estimate
11:         $\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$     // Compute bias-corrected second raw moment estimate
12:         $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}$     // Update parameters
13:     end while
14:     return $\boldsymbol{\theta}_t$         // Resulting parameters
15: end function
```

class	count	width / pixel	height / pixel
not relevant	382.027	16 ± 13	36 ± 25
relevant	294.776	24 ± 18	55 ± 32
not relevant, occluded	81.549	18 ± 17	36 ± 26
relevant, occluded	33.196	32 ± 29	52 ± 34
horizontal	454	31 ± 21	13 ± 7
vertical	746.055	20 ± 17	43 ± 29
horizontal bus	40	32 ± 22	18 ± 4
vertical bus	44.999	23 ± 18	57 ± 37
one light	18.126	19 ± 16	14 ± 10
two lights	142.930	16 ± 14	29 ± 19
three lights	613.207	21 ± 18	47 ± 30
four lights	17.285	22 ± 18	68 ± 40
off	178.530	17 ± 15	35 ± 22
red	231.109	20 ± 16	44 ± 31
yellow	27.398	23 ± 18	47 ± 33
red-yellow	11.796	23 ± 20	52 ± 35
green	342.715	22 ± 18	48 ± 32
circle	544.911	21 ± 18	46 ± 31
arrow straight	30.112	24 ± 20	54 ± 33
arrow left	57.020	23 ± 18	51 ± 31
arrow straight left	50	22 ± 17	54 ± 33
arrow right	6.831	27 ± 20	56 ± 32
pedestrian	132.703	15 ± 11	31 ± 19
cyclist	19.921	18 ± 15	43 ± 29
total	791.548	20 ± 17	44 ± 30

Tab. 14: Frequency of the individual classes in the complete DriveU dataset together with the mean and standard deviation of pixel width and height. The total number of annotations is 791.548.

drop rate	learning rate	weight decay	loss	f-score	f-score:state	f-score:type
0.2	0.10000	1.0000	0.013747	0.411427	0.266899	0.555956
0.1	1.00000	1.0000	0.013535	0.411427	0.266899	0.555956
0.0	0.01000	1.0000	0.011997	0.411427	0.266899	0.555956
0.5	1.00000	0.0001	0.011243	0.418438	0.266899	0.569977
0.3	0.10000	0.0100	0.009962	0.551987	0.553406	0.554185
0.2	0.10000	0.0010	0.008621	0.651777	0.710733	0.592821
0.2	0.00100	0.1000	0.008242	0.676197	0.809581	0.584866
0.4	0.01000	0.0100	0.008015	0.678898	0.780265	0.577530
0.3	0.00100	0.1000	0.008437	0.679034	0.808666	0.581890
0.1	0.10000	0.0001	0.007239	0.682086	0.865650	0.584311
0.1	0.00100	0.1000	0.008712	0.697817	0.828187	0.587999
0.4	0.00100	0.1000	0.007679	0.709288	0.846182	0.582265
0.5	0.01000	0.0100	0.006948	0.729624	0.856527	0.626894
0.1	0.01000	0.0100	0.007099	0.731544	0.867693	0.595396
0.2	0.01000	0.0100	0.006849	0.733291	0.867631	0.632753
0.3	0.01000	0.0100	0.006884	0.740068	0.855668	0.630197
0.3	0.00001	1.0000	0.006098	0.757355	0.870155	0.651030
0.4	0.00001	0.0010	0.006043	0.782986	0.910453	0.658022
0.4	0.00001	0.0100	0.005374	0.789602	0.916865	0.669974
0.3	0.00001	0.0100	0.005263	0.801207	0.918563	0.690420
0.4	0.00001	0.1000	0.005352	0.801416	0.918594	0.688073
0.4	0.01000	0.0010	0.005504	0.805990	0.908413	0.708626
0.1	0.01000	0.0010	0.005570	0.806880	0.910934	0.720384
0.2	0.00001	1.0000	0.005483	0.808588	0.896149	0.721251
0.3	0.01000	0.0010	0.005229	0.809000	0.909295	0.715693
0.1	0.00001	1.0000	0.005321	0.812541	0.895308	0.733576
0.3	0.00001	0.0010	0.005081	0.815871	0.920530	0.711222
0.2	0.01000	0.0010	0.005286	0.817727	0.910526	0.733965
0.3	0.00001	0.1000	0.004750	0.826267	0.919485	0.735642
0.2	0.00001	0.0010	0.004619	0.826948	0.923089	0.734061
0.1	0.00001	0.0001	0.004509	0.838738	0.923205	0.755479
0.2	0.00001	0.1000	0.004585	0.839313	0.925173	0.755911
0.2	0.00001	0.0100	0.004569	0.840202	0.923132	0.757393
0.1	0.00001	0.1000	0.004304	0.847056	0.926504	0.770663
0.1	0.00001	0.0100	0.004254	0.850246	0.933185	0.768545
0.4	0.01000	0.0001	0.003778	0.857369	0.931416	0.791537
0.4	0.00100	0.0100	0.004108	0.864443	0.932296	0.798393
0.3	0.01000	0.0001	0.003729	0.869562	0.934022	0.805102
0.3	0.00100	0.0100	0.003881	0.871481	0.930866	0.822456
0.2	0.01000	0.0001	0.003652	0.874602	0.938411	0.815406
0.2	0.00100	0.0100	0.003707	0.874693	0.938491	0.821777
0.1	0.01000	0.0001	0.003400	0.878737	0.940165	0.822704
0.4	0.00010	0.0001	0.004055	0.878754	0.936522	0.826125
0.3	0.00010	0.0001	0.003632	0.883433	0.942895	0.830372
0.2	0.00010	0.0001	0.003581	0.883559	0.942790	0.828630
0.1	0.00100	0.0100	0.003433	0.885458	0.937963	0.838489
0.4	0.00010	0.1000	0.003629	0.885561	0.939575	0.846155
0.3	0.00010	0.0010	0.003496	0.887890	0.945977	0.834068
0.1	0.00010	0.0010	0.003723	0.888153	0.941191	0.840202
0.4	0.00010	0.0010	0.003736	0.888562	0.943279	0.835735
0.2	0.00010	0.0010	0.003518	0.888661	0.944959	0.835288
0.1	0.00010	0.0001	0.003569	0.891797	0.941053	0.845450
0.3	0.00010	0.1000	0.003580	0.892228	0.940617	0.847457
0.2	0.00010	0.1000	0.003397	0.894523	0.943268	0.848972
0.1	0.00010	0.0100	0.003273	0.898869	0.943970	0.861706
0.3	0.00100	0.0001	0.003092	0.902453	0.951906	0.855918
0.3	0.00100	0.0010	0.002952	0.903246	0.947744	0.865282
0.2	0.00100	0.0010	**0.002874**	0.903486	0.949044	0.867450
0.1	0.00100	0.0001	0.003233	0.903529	0.947532	0.860949
0.1	0.00100	0.0010	0.002928	0.905328	0.946176	0.866719
0.4	0.00100	0.0001	0.003084	0.905509	**0.952772**	0.860998
0.4	0.00010	0.0100	0.003352	0.907494	0.949557	0.868068
0.4	0.00100	0.0010	0.002938	0.907804	0.951338	0.867748
0.2	0.00100	0.0001	0.003150	0.907942	0.950693	0.872517
0.2	0.00010	0.0100	0.003085	0.909787	0.952077	0.873746
0.3	0.00010	0.0100	0.003049	**0.911185**	0.949691	**0.874437**

Tab. 15: Results of the random search on the hyperparameters of the type and state classifier together with the results of the subsequent *PathSearch* algorithm. One row corresponds to a drop rate, learning rate and weight decay configuration. The loss and the f-scores are calculated from the evaluation on the validation set that is performed after every training epoch, with the denoted value forming the best score on the respective configuration. Here the f-score is the mean overall class individual f-scores weighted by the positive samples in that class. The rows are sorted ascending concerning the f-score and the bold font indicates the best score in a column.

drop rate	depth	learning rate	weight decay	loss	f-score
0.1	18	1e-04	1e-06	1.345e-03	0.000000
0.1	15	1e-03	1e-01	1.266e-03	0.000000
0.3	7	1e-01	1e+00	1.357e-03	0.000000
0.5	6	1e-05	1e+00	1.344e-03	0.000000
0.0	9	1e-03	1e+00	1.358e-03	0.000000
0.2	11	1e+00	1e+00	1.347e-03	0.000000
0.5	7	1e-01	1e+00	1.357e-03	0.000000
0.3	12	1e-04	1e+00	1.348e-03	0.000000
0.5	12	1e-05	1e-01	1.347e-03	0.599072
0.5	16	1e-05	1e-03	1.345e-03	0.599072
0.5	15	1e+00	1e-02	1.345e-03	0.599072
0.1	16	1e-03	1e-01	1.142e-03	0.765769
0.1	17	1e-03	1e-01	1.102e-03	0.798062
0.1	16	1e-02	1e-03	8.848e-04	0.800807
0.1	17	1e-02	1e-06	8.954e-04	0.803359
0.1	16	1e-02	1e-05	8.669e-04	0.806684
0.1	18	1e-02	1e-06	8.317e-04	0.806698
0.1	15	1e-02	1e-03	9.286e-04	0.808028
0.3	11	1e-03	1e-01	1.068e-03	0.813899
0.1	16	1e-02	1e-06	8.440e-04	0.821416
0.1	14	1e-03	1e-04	8.093e-04	0.829198
0.0	16	1e-03	1e-01	8.760e-04	0.832617
0.1	14	1e-03	1e-03	8.010e-04	0.832676
0.5	6	1e-01	1e-05	8.562e-04	0.834115
0.1	18	1e-03	1e-05	7.942e-04	0.834267
0.0	13	1e-03	1e-01	9.451e-04	0.836773
0.1	18	1e-02	1e-04	7.432e-04	0.837335
0.0	17	1e-03	1e-01	8.675e-04	0.837463
0.0	14	1e-02	1e-03	6.937e-04	0.838249
0.2	11	1e-01	1e-05	6.852e-04	0.838651
0.0	15	1e-03	1e-01	8.585e-04	0.838807
0.1	18	1e-04	1e-05	7.360e-04	0.839065
0.1	14	1e-01	1e-05	7.157e-04	0.839333
0.0	15	1e-03	1e-02	6.956e-04	0.839906
0.2	10	1e-02	1e-02	7.110e-04	0.840266
0.0	15	1e-02	1e-03	6.760e-04	0.840342
0.1	14	1e-02	1e-03	6.906e-04	0.840386
0.1	17	1e-04	1e-01	9.711e-04	0.840858
0.1	17	1e-05	1e-04	6.330e-04	0.841290
0.1	16	1e-04	1e-01	9.461e-04	0.841426
0.1	16	1e-02	1e-04	6.916e-04	0.841459
0.1	17	1e-03	1e-05	6.606e-04	0.841686
0.0	16	1e-02	1e-03	6.611e-04	0.841796
0.1	15	1e-03	1e-02	7.102e-04	0.842046
0.1	17	1e-03	1e-02	6.974e-04	0.842427
0.1	15	1e-04	1e-01	9.669e-04	0.842571
0.0	16	1e-05	1e-03	6.266e-04	0.842675
0.1	15	1e-02	1e-04	6.716e-04	0.843237
0.1	17	1e-02	1e-05	6.884e-04	0.843243
0.1	15	1e-03	1e-04	6.279e-04	0.843264
0.1	16	1e-03	1e-02	7.408e-04	0.843317
0.1	15	1e-02	1e-06	6.696e-04	0.843811
0.1	15	1e-02	1e-05	6.754e-04	0.843999
0.1	16	1e-05	1e-04	7.060e-04	0.844476
0.1	14	1e-02	1e-04	6.767e-04	0.844617
0.1	14	1e-04	1e-06	7.375e-04	0.844733
0.1	15	1e-05	1e-02	7.076e-04	0.844772
0.0	15	1e-05	1e-03	6.559e-04	0.844895
0.1	17	1e-02	1e-04	6.684e-04	0.845203
0.1	18	1e-02	1e-05	7.049e-04	0.845330
0.1	16	1e-02	1e-07	6.890e-04	0.845437
0.1	13	1e-02	1e-05	7.002e-04	0.845817
0.0	14	1e-02	1e-04	6.307e-04	0.846212
0.1	14	1e-03	1e-07	6.422e-04	0.846285
0.1	13	1e-02	1e-07	6.416e-04	0.846362
0.1	17	1e-05	1e-02	6.667e-04	0.846421
0.0	12	1e-02	1e-04	6.283e-04	0.846460
0.0	18	1e-02	1e-04	6.296e-04	0.846525
0.0	16	1e-03	1e-02	6.542e-04	0.846693
0.0	15	1e-02	1e-04	6.277e-04	0.846847
0.0	16	1e-05	1e-02	6.761e-04	0.846891
0.0	17	1e-05	1e-02	6.706e-04	0.847006
0.0	17	1e-05	1e-04	6.437e-04	0.847101
0.0	15	1e-05	1e-02	6.778e-04	0.847314
0.1	17	1e-03	1e-03	6.600e-04	0.847372
0.1	15	1e-05	1e-04	6.348e-04	0.847377
0.0	17	1e-04	1e-01	8.262e-04	0.847490
0.0	16	1e-04	1e-01	8.293e-04	0.847962
0.1	15	1e-05	1e-03	6.780e-04	0.847971
0.1	15	1e-04	1e-04	6.899e-04	0.848006
0.0	16	1e-02	1e-04	6.281e-04	0.848006
0.0	16	1e-05	1e-04	6.445e-04	0.848082
0.1	13	1e-03	1e-07	6.939e-04	0.848091
0.1	15	1e-04	1e-07	6.391e-04	0.848179
0.0	15	1e-05	1e-04	6.140e-04	0.848234
0.0	17	1e-05	1e-03	6.378e-04	0.848360
0.0	17	1e-02	1e-04	6.300e-04	0.848363
0.1	17	1e-05	1e-03	6.644e-04	0.848643
0.1	14	1e-03	1e-05	6.688e-04	0.848863
0.0	15	1e-04	1e-01	8.411e-04	0.848886
0.0	17	1e-03	1e-02	6.502e-04	0.848986
0.1	16	1e-03	1e-03	6.795e-04	0.848989
0.1	17	1e-04	1e-03	6.287e-04	0.849387
0.2	16	1e-05	1e-03	6.780e-04	0.849545
0.0	16	1e-02	1e-05	6.187e-04	0.849566
0.1	14	1e-04	1e-04	6.317e-04	0.849723
0.1	17	1e-04	1e-02	7.509e-04	0.849909
0.0	15	1e-04	1e-07	6.046e-04	0.850136
0.1	14	1e-02	1e-05	6.236e-04	0.850145
0.1	13	1e-04	1e-05	6.432e-04	0.850299
0.1	16	1e-05	1e-01	8.988e-04	0.850299
0.1	17	1e-05	1e-01	9.366e-04	0.850313
0.1	15	1e-05	1e-01	1.019e-03	0.850400
0.1	15	1e-03	1e-03	6.309e-04	0.850500
0.0	16	1e-04	1e-06	6.028e-04	0.850502
0.1	14	1e-03	1e-06	6.558e-04	0.850638
0.0	18	1e-04	1e-06	6.021e-04	0.850667
0.0	14	1e-04	1e-06	6.079e-04	0.850721
0.1	12	1e-05	1e-05	6.315e-04	0.850878
0.1	16	1e-05	1e-03	6.715e-04	0.850991

drop rate	depth	learning rate	weight decay	loss	f-score
0.1	16	1e-04	1e-03	6.254e-04	0.851008
0.0	15	1e-02	1e-05	6.150e-04	0.851012
0.1	16	1e-03	1e-04	6.573e-04	0.851020
0.0	15	1e-04	1e-06	5.986e-04	0.851187
0.0	13	1e-04	1e-05	6.009e-04	0.851205
0.0	13	1e-02	1e-05	6.126e-04	0.851224
0.0	18	1e-02	1e-05	6.142e-04	0.851238
0.0	14	1e-04	1e-07	6.021e-04	0.851338
0.0	14	1e-03	1e-03	6.222e-04	0.851442
0.1	16	1e-04	1e-07	6.411e-04	0.851537
0.1	17	1e-03	1e-06	6.320e-04	0.851643
0.0	16	1e-05	1e-01	7.976e-04	0.851660
0.0	13	1e-04	1e-06	5.994e-04	0.851665
0.1	15	1e-03	1e-06	6.316e-04	0.851792
0.0	16	1e-04	1e-05	6.007e-04	0.851813
0.1	16	1e-04	1e-02	6.527e-04	0.851828
0.0	17	1e-03	1e-03	6.060e-04	0.851870
0.0	17	1e-02	1e-05	6.055e-04	0.851925
0.0	14	1e-02	1e-05	6.048e-04	0.852116
0.1	15	1e-02	1e-07	6.214e-04	0.852154
0.0	17	1e-04	1e-06	5.973e-04	0.852158
0.0	16	1e-03	1e-03	6.108e-04	0.852184
0.1	14	1e-02	1e-06	6.235e-04	0.852300
0.0	15	1e-03	1e-03	6.059e-04	0.852300
0.1	16	1e-04	1e-06	6.252e-04	0.852343
0.0	16	1e-04	1e-04	5.996e-04	0.852345
0.0	14	1e-04	1e-05	5.983e-04	0.852362
0.1	16	1e-03	1e-06	6.491e-04	0.852383
0.0	17	1e-04	1e-05	5.973e-04	0.852395
0.0	18	1e-04	1e-04	5.970e-04	0.852401
0.0	15	1e-04	1e-03	5.947e-04	0.852403
0.0	14	1e-04	1e-03	5.942e-04	0.852477
0.1	13	1e-03	1e-05	6.176e-04	0.852491
0.1	17	1e-03	1e-04	6.192e-04	0.852503
0.0	16	1e-04	1e-03	5.956e-04	0.852645
0.1	18	1e-03	1e-04	6.486e-04	0.852654
0.0	15	1e-05	1e-01	8.109e-04	0.852698
0.0	16	1e-04	1e-07	5.986e-04	0.852720
0.1	15	1e-04	1e-02	6.654e-04	0.852724
0.1	18	1e-04	1e-04	6.264e-04	0.852791
0.0	18	1e-04	1e-05	5.963e-04	0.852813
0.0	13	1e-04	1e-07	5.978e-04	0.852843
0.1	13	1e-02	1e-06	6.175e-04	0.852912
0.0	15	1e-04	1e-04	5.980e-04	0.852945
0.1	13	1e-04	1e-07	6.104e-04	0.852971
0.0	17	1e-04	1e-03	5.959e-04	0.853001
0.0	17	1e-04	1e-04	5.998e-04	0.853016
0.1	16	1e-05	1e-02	6.516e-04	0.853057
0.0	15	1e-04	1e-05	5.997e-04	0.853111
0.1	13	1e-04	1e-06	6.127e-04	0.853259
0.0	14	1e-04	1e-04	6.083e-04	0.853279
0.1	16	1e-04	1e-04	6.251e-04	0.853331
0.1	15	1e-04	1e-05	6.153e-04	0.853340
0.0	17	1e-05	1e-01	8.024e-04	0.853481
0.0	13	1e-02	1e-06	5.952e-04	0.853646
0.1	15	1e-04	1e-03	6.364e-04	0.853714
0.1	17	1e-04	1e-06	6.246e-04	0.853777
0.1	17	1e-04	1e-04	6.170e-04	0.853791
0.1	16	1e-04	1e-05	6.326e-04	0.853836
0.1	16	1e-03	1e-07	6.046e-04	0.853926
0.0	16	1e-03	1e-07	6.011e-04	0.853956
0.1	14	1e-02	1e-07	6.126e-04	0.854008
0.0	13	1e-03	1e-05	5.874e-04	0.854215
0.1	15	1e-03	1e-07	6.034e-04	0.854221
0.0	17	1e-04	1e-02	6.191e-04	0.854303
0.0	15	1e-02	1e-06	5.953e-04	0.854348
0.1	14	1e-04	1e-05	6.093e-04	0.854402
0.1	14	1e-04	1e-03	6.143e-04	0.854484
0.1	13	1e-03	1e-06	6.049e-04	0.854528
0.1	16	1e-03	1e-05	6.048e-04	0.854734
0.0	13	1e-03	1e-07	5.862e-04	0.854839
0.1	18	1e-03	1e-06	6.188e-04	0.854898
0.1	17	1e-04	1e-05	6.108e-04	0.854924
0.1	15	1e-03	1e-05	6.117e-04	0.854976
0.0	15	1e-04	1e-02	5.991e-04	0.855026
0.0	17	1e-03	1e-06	5.850e-04	0.855038
0.1	15	1e-04	1e-06	6.142e-04	0.855099
0.0	16	1e-03	1e-06	5.851e-04	0.855127
0.0	14	1e-03	1e-07	5.853e-04	0.855210
0.0	14	1e-02	1e-07	5.946e-04	0.855309
0.1	10	1e-05	1e-02	6.072e-04	0.855439
0.1	14	1e-04	1e-07	6.388e-04	0.855449
0.0	16	1e-02	1e-06	5.875e-04	0.855643
0.0	18	1e-03	1e-05	5.837e-04	0.855751
0.0	18	1e-03	1e-06	5.833e-04	0.855844
0.0	17	1e-02	1e-06	5.924e-04	0.855850
0.0	16	1e-04	1e-02	5.958e-04	0.855891
0.0	18	1e-02	1e-06	5.925e-04	0.856033
0.0	13	1e-02	1e-07	5.894e-04	0.856114
0.0	17	1e-03	1e-04	5.886e-04	0.856320
0.0	15	1e-03	1e-05	5.871e-04	0.856373
0.0	15	1e-03	1e-06	5.822e-04	0.856462
0.0	15	1e-03	1e-04	5.853e-04	0.856470
0.0	14	1e-02	1e-06	5.886e-04	0.856531
0.0	15	1e-03	1e-07	5.841e-04	0.856533
0.0	13	1e-03	1e-06	5.868e-04	0.856591
0.0	15	1e-02	1e-07	5.891e-04	0.856662
0.0	14	1e-03	1e-06	**5.805e-04**	0.856814
0.0	18	1e-03	1e-04	5.867e-04	0.856859
0.0	16	1e-03	1e-04	5.843e-04	0.856892
0.0	16	1e-03	1e-05	5.845e-04	0.857073
0.0	16	1e-02	1e-07	5.850e-04	0.857131
0.0	14	1e-03	1e-05	5.839e-04	0.857278
0.0	14	1e-03	1e-04	5.867e-04	0.857574
0.0	17	1e-03	1e-05	5.826e-04	**0.858465**

Tab. 15: Results of the random search on the hyperparameters of the relevance classifier together with the results of the subsequent *PathSearch* algorithm. One row corresponds to a drop rate, depth, learning rate and weight decay configuration. The loss and the f-scores are calculated from the evaluation on the validation set that is performed after every training epoch, with the denoted value forming the best score on the respective configuration. Here the rows are sorted ascending concerning the f-score and the bold font indicates the best score in a column.

Label	Precision	Recall	F-Score	Accuracy	Correct	Incorrect	N.o.p.s.	N.o.n.s.
off	0.946978	0.899652	0.922709	0.962033	231.469	9.135	60.609	179.995
red	0.940421	0.936146	0.938279	0.965425	232.285	8.319	67.545	173.059
yellow	0.739259	0.862909	0.796312	0.986135	237.268	3.336	7.557	233.047
red-yellow	0.722723	0.697557	0.709917	0.991563	238.574	2.030	3.561	237.043
green	0.959170	0.979957	0.969452	0.973990	234.346	6.258	101.332	139.272
circle	0.932304	0.914689	0.923412	0.896165	215.621	24.983	164.656	75.948
arrow straight	0.815767	0.769604	0.792013	0.985453	237.104	3.500	8.659	231.945
arrow left	0.829640	0.874914	0.851676	0.977835	235.271	5.333	17.500	223.104
arrow straight left	1.000000	0.000000	0.000000	0.999967	240.596	8	8	240.596
arrow right	0.703022	0.682714	0.692719	0.994510	239.283	1.321	2.181	238.423
pedestrian	0.739319	0.816948	0.776197	0.919498	221.235	19.369	41.114	199.490
cyclist	0.598649	0.450971	0.514421	0.977049	235.082	5.522	6.486	234.118
relevant	0.602843	0.313159	0.412195	0.628988	151.337	89.267	99.946	140.658

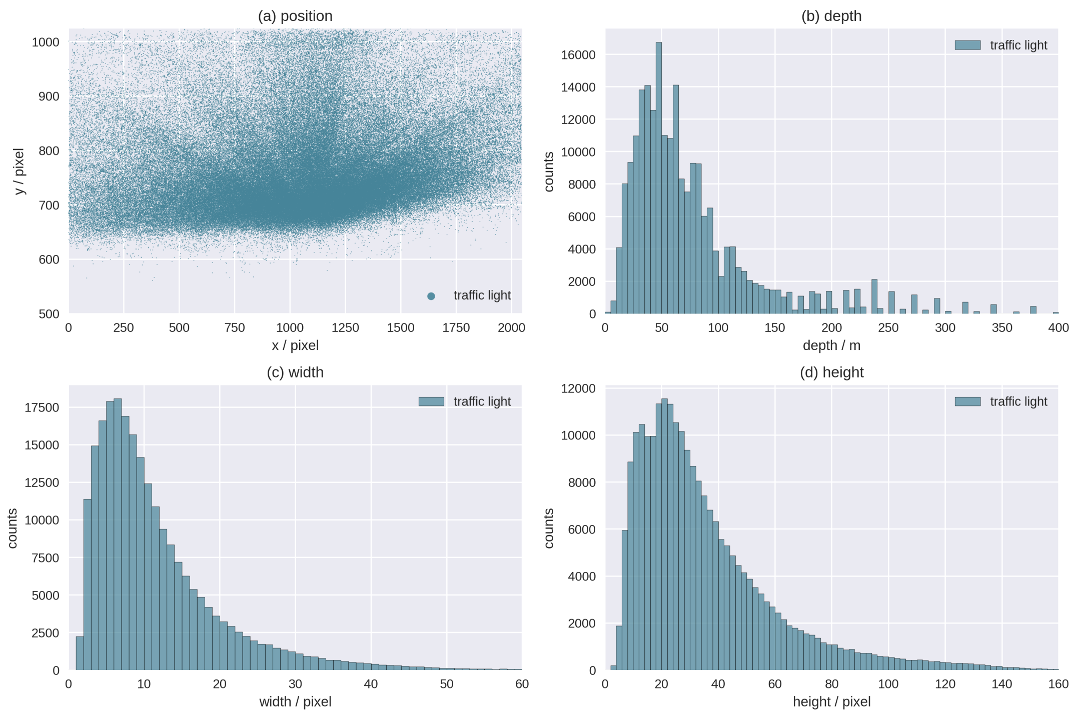
Tab. 16: The results on the thirteen (fourteen) labels of the combined classification for the reduced DriveUC test set. The total number of samples in the DriveUC test set is 240.604. We omitted the "not relevant" label since the information about it is redundant with the "relevant" label. N.o.p.s. and N.o.n.s. stand for the Number of positive/negative samples respectively.

DriveU error	C(P=yes, S=red)	C(P=yes, S=red-yellow)	C(P=yes, S=yellow)	C(P=yes, S=green)	C(P=no, S=red)	C(P=no, S=red-yellow)	C(P=no, S=yellow)	C(P=no, S=green)
segmentation	7	2	7	53	2	0	0	0
traffic light classification	0	0	0	0	0	5	1	0
identification	0	0	0	7	7	7	5	2
total number of samples	100	3	8	67	100	100	100	100

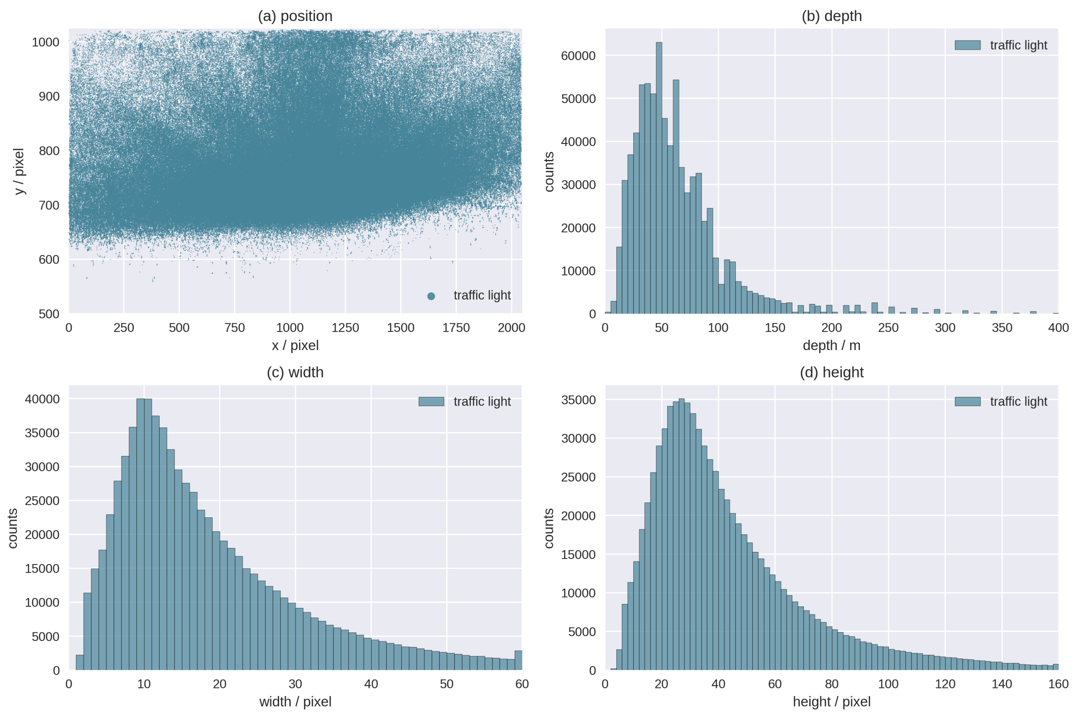
Cityscapes train error	C(P=yes, S=red)	C(P=yes, S=red-yellow)	C(P=yes, S=yellow)	C(P=yes, S=green)	C(P=no, S=red)	C(P=no, S=red-yellow)	C(P=no, S=yellow)	C(P=no, S=green)
segmentation	0	0	0	0	0	0	0	0
traffic light classification	1	0	0	0	12	0	1	5
identification	1	0	0	0	15	0	0	11
total number of samples	43	0	1	2	100	4	6	100

Cityscapes train-extra, test error	C(P=yes, S=red)	C(P=yes, S=red-yellow)	C(P=yes, S=yellow)	C(P=yes, S=green)	C(P=no, S=red)	C(P=no, S=red-yellow)	C(P=no, S=yellow)	C(P=no, S=green)
segmentation	1	0	0	2	1	0	1	1
traffic light classification	1	0	1	1	8	1	3	4
identification	0	0	0	3	16	6	2	3
total number of samples	53	0	1	8	100	19	41	100

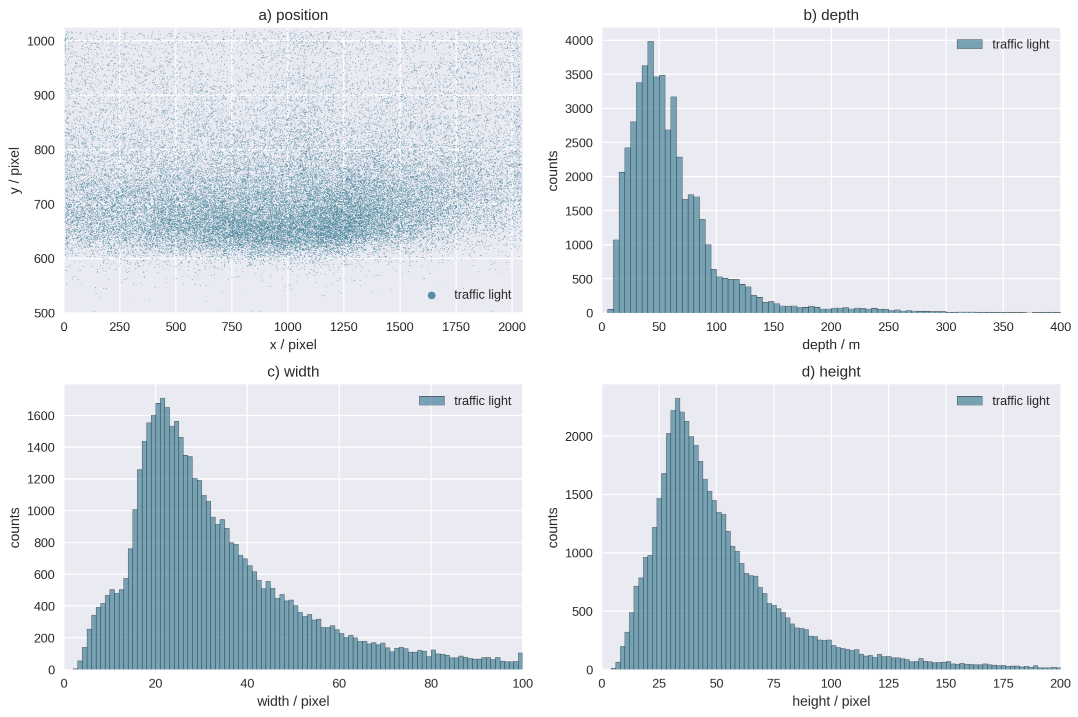
Tab. 17: The error counts in the different categories made by the PC identification and pedestrian localisation algorithm for the used datasets. We either observe all samples in a category, or just 100 samples if the number of samples in a category exceeds this limit.



(a) DriveU



(b) DriveUC



(c) Cityscapes

Fig. 23: The depth, position, width and height features of the traffic light bounding boxes for the DriveU, DriveUC and Cityscapes datasets. Notice, that the x-axis in the width and the height diagram of Cityscapes cover a bigger range than the ones of DriveuU and DriveUC.

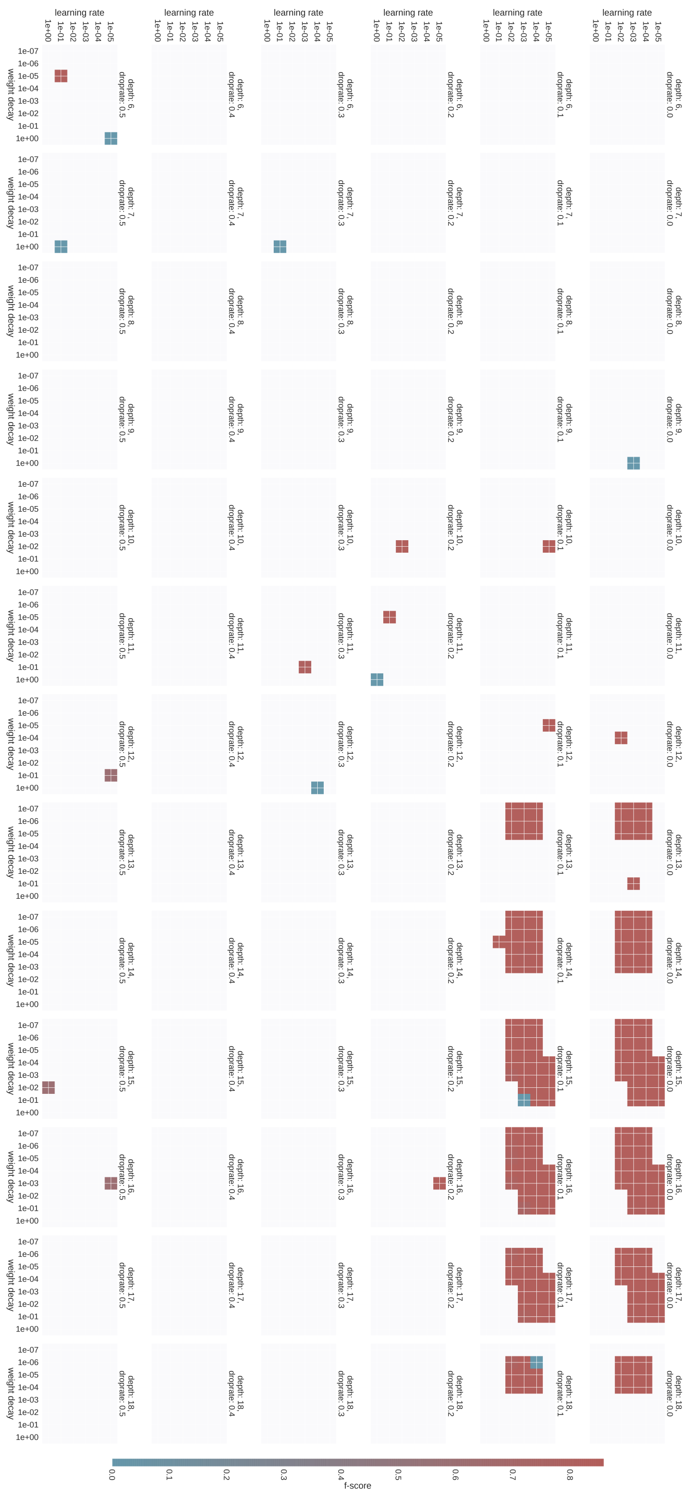
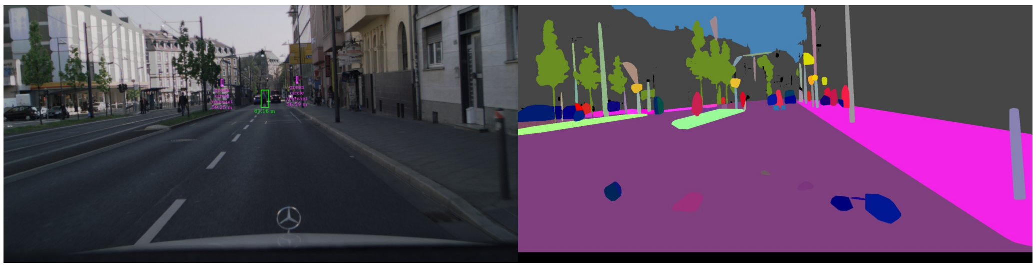
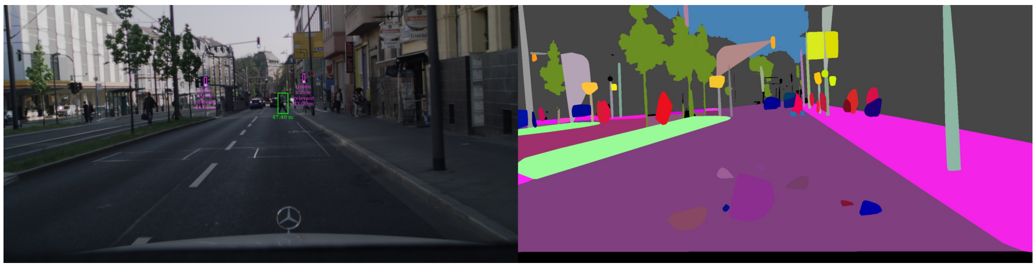


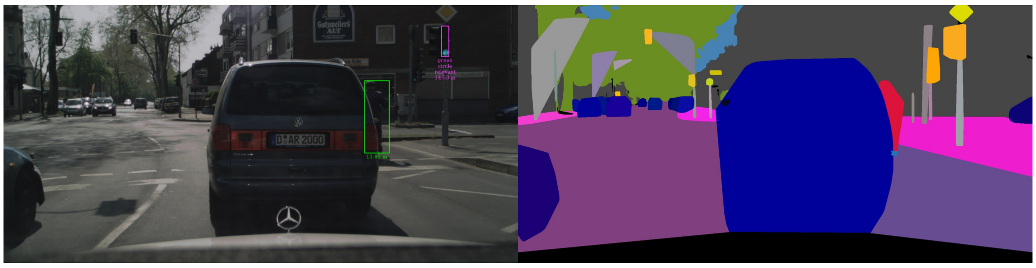
Fig. 24: The random path search for the hyperparameter of the relevance classifier. Every patch (i.e. cross) corresponds to a certain configuration of depth, drop rate, learning rate and weight decay. The colour of the patch indicates the best validation f-score on the respective configuration. The white patches correspond to configurations that were not explored.

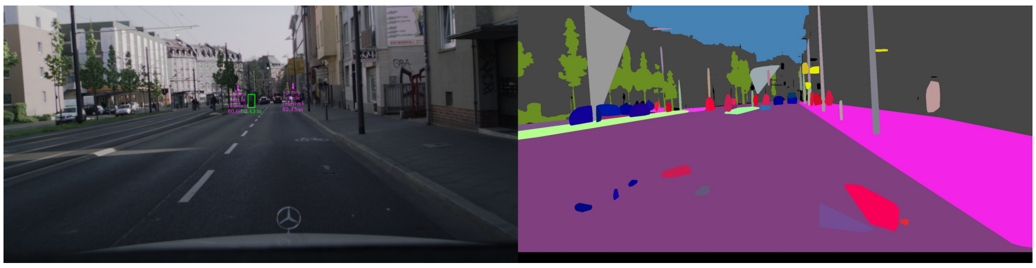

(a) DriveU

(b) DriveU

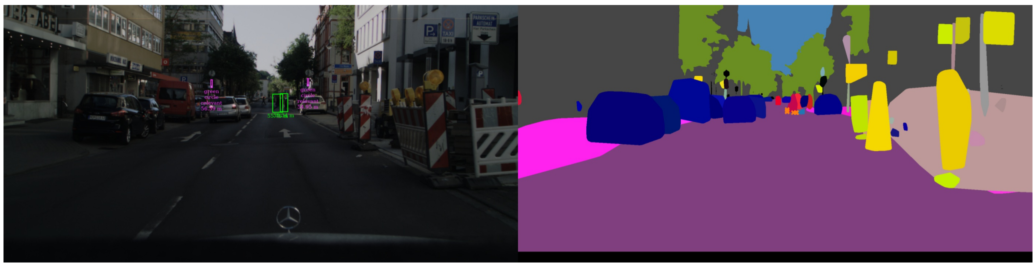
(c) DriveU


(d) DriveU

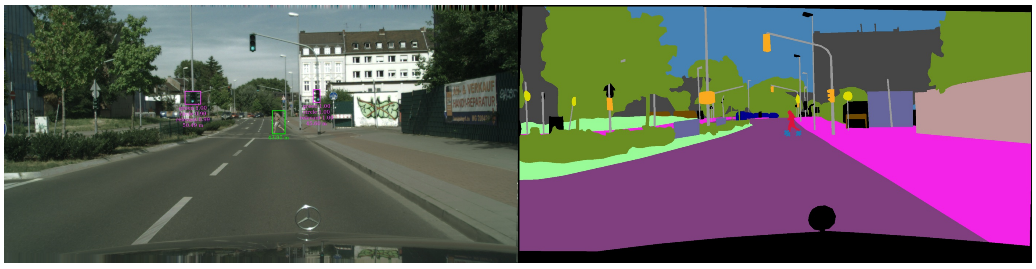

(e) DriveU



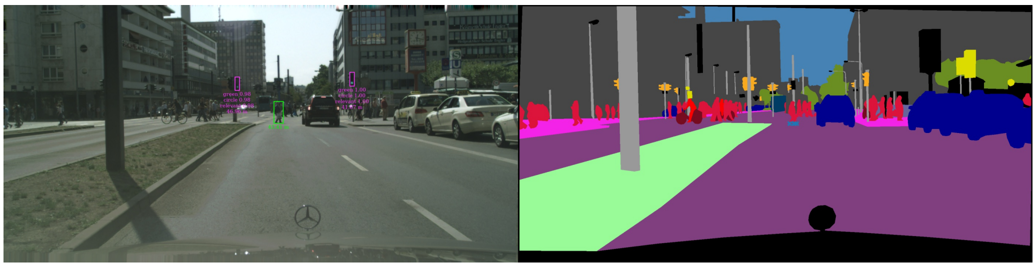
(f) DriveU



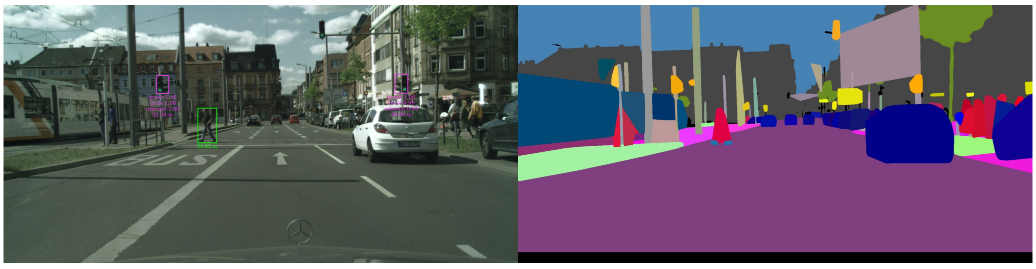
(g) DriveU



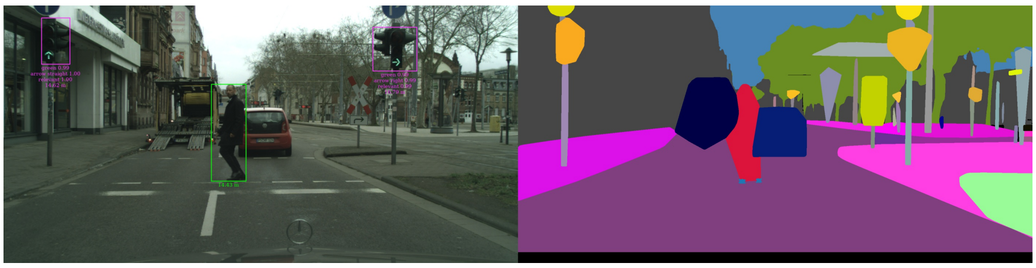
(h) Cityscapes train



(i) Cityscapes train



(j) Cityscapes train-extra, test



(k) Cityscapes train-extra, test

Fig. 25: All the P(P=yes, state=green) samples from the examined datasets, after excluding wrong samples. The images contain bounding boxes around the managing traffic lights and the detected pedestrian, together with the depth information and for Cityscapes the output probabilities of the classifier. We also added the panoptic segmentation with the points used to make up the feet-polygon of the detected pedestrian.

Erklärung

Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den September 21, 2020,