UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Departamento de Sistemas, Informáticos y Computación

Universitat Politècnica de València

Real Time Semantic Urban Scene Segmentation and Lane Recognition

Master's Thesis

Máster Universitario en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

AUTOR	Janik Jilka
TUTOR	Roberto Paredes Palacios
EXPERIMENTAL DIRECTOR	Juan Maroñas Molano

Academic year 2020-2021

Acknowledgements

Throughout the writing of this master's thesis and through the whole master's studies I have received a great deal of support and assistance.

First of all I want to thank my supervisor Professor R. Paredes Palacios whose expertise was invaluable in formulating the research questions and methodology. Your insightful feedback pushed me to sharpen my thinking and brought my work to a higher level.
Same I want to give my thank to my tutor J. Maroñas Molano who has always helped me through his valuable guidance throughout this work. Especially by providing me the tools and working environment needed to complete my research.

As well I want to thank Bosch for allowing me to participate in this master's program and gather all these experiences and know-how.

I also want to thank my colleagues from this master studies, above all J. Cuevas Muñoz. It has been great cooperation and assistance throughout the whole studies. Most of all, I would like to thank my family and friends for their constant support. You are always there for me to guide me through this experience in an unfamiliar environment.

Abstract

Perception systems are the groundwork for all systems of self-driving cars. These need to detect the environment and the driving scene to provide every information needed to driving systems to solve every situation.
As perception systems, this work focuses on the usage of convolutional neural networks to process and analyze camera input data.
This work presents an in-depth scene understanding consisting of object detection and a lane detection model. The object detection is realized as a semantic segmentation of the input map. A pixel-wise classification returns all information about the scene.
The lane detection is based on a line detection model. The driving lane is concluded by detecting the street lines and the road. This part combines the results with the ones obtained by the segmentation. This returns a comprehensive understanding of the current traffic scene.
Therefore this work compares different network structures and approaches to solve this problem. An evaluation presents the best model suited for the usage in autonomous driving vehicles.

Resumen

Los sistemas de percepción son la base para todos los sistemas de conducción autónoma. Estos detectan el ambiente y la escena de conducción para proveer la información necesaria para tomar decisiones frente a cada situación.
Este trabajo se enfoca en el uso de redes neuronales convolucionales para procesar y analizar los datos de entrada de la cámara.
El presente trabajo exhibe una compresión profunda de la escena captada, basándose en modelos de detección de objetos y detección de los carriles vehiculares. La detección de objetos se realiza como una segmentación semántica del mapa de entrada, es decir, una clasificación por píxeles la cual devuelve toda la información sobre la escena.
La detección de carriles vehiculares se basa en un modelo de detección de líneas. El carrill se calcula detectando las líneas de la calle y la forma de la carretera. Lo anterior combinado con los resultados obtenidos por la segmentación, devuelve una comprensión completa de la escena del tráfico actual.
Por otra parte, este trabajo compara diferentes estructuras de redes neuronales convolucionales y enfoques para resolver el problema planteado. Finalmente, basado en las pruebas realizadas, se propone el mejor modelo para la utilización en vehículos de conducción autónoma.

Acronyms

ACC Adaptive Cruise Control. 8, 9

ADAS Advanced Driver Assistance Systems. 1

CE Cross Entropy. 67, 69, 70

CNN Convolutional Neural Network. 9, 19–21, 30

CRF Conditional Random Field. 16

FC Fully Connected. 15, 20

FCN Fully Convolutional Network. 15, 16, 32

FCW Forward Collision Warning. 8, 9

FFM Feature Fusion Module. 29

FIR Finite Impulse Response. 20

FN False Negatives. 11

FP False Positives. 11, 12, 73, 74, 85

fps frames per second. xiv, 10, 16, 18, 82

IoU Intersection over Union. xi, 12, 13, 15, 17–19, 29, 32, 35, 40–43, 51, 52, 54–58, 60, 62, 68

LDW Lane Departure Warning. 8, 9

PPM Pyramid Pooling Module. 17, 18, 36, 38

PSP Pyramid Scene Parsing. 17

ReLU Rectified Linear Unit. 31, 33

ROI Region of Interest. 2, 20

SAE Society of Automotive Engineers. 1

SCNN Segmentation Convolutional Neural Network. 18, 25, 27

SDC Self Driving Cars. ix, 8, 87

SGD Stochasitc Gradient Descent. 82

TN True Negatives. 11, 83

TP True Positives. 11, 73

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Aim of Work

The development of self-driving systems has been under research and in development for lots of years. Now, autonomous vehicles with these functions are already present in the traffic. Still, most systems require a driver to supervise these systems because there still exist situations that cannot be solved by a rule-based technical system. Therefore the institution Society of Automotive Engineers (SAE) defined five different levels of automation. Above level 3 it usually is called autonomous driving. The different levels provide more functions and systems that assist the driver up to full automation. [1]

Key factors for achieving full automation are the interaction design of automated vehicles and a better scene understanding. Situations that cannot be solved by the system on its own are often due to insufficient information about the environment. A better scene understanding returns more information to the different functions which enable a better driving behavior and fewer driver interventions are required. Unknown situations can still be solved by providing enough information to the system. Therefore the perception part is the building block of the whole system and all other functions depend on its performance.

By designing the perception system of modern vehicles mainly the following sensor types are used: cameras, radars, or LiDAR. When tied to a computing system, each sensor can support the Advanced Driver Assistance Systems (ADAS) that allow a vehicle to operate autonomously in an environment. Due to modern computer vision technology cameras are the most promising system for future systems.

1.2 Problem of Work

This work tackles the problem of designing a perception system for self-driving cars. Therefore camera data is used to detect and provide as much information as possible.

The understanding of street scenes limits the group of relevant objects which need to be detected due to the purpose of their interference behavior. In general, there can be defined the following relevant groups of objects:

- limiting driving area (road, lane, border area)
- traffic participants (different types of vehicles, pedestrians, other dynamic objects.)
- traffic control objects (traffic signs, traffic lights, etc.)
- obstacles (static objects)

First of all the driving area need to be defined. Therefore the road and especially the lane need to be detected. These define the Region of Interest (ROI) of these self-driving systems.
Next all objects which can appear in this area need to be detected. The obstacle is not only detecting them but also classifying them to estimate their behavior. Some groups like persons need more attention than others. As well, it is important to distinguish between cars, bicycles or trains, etc because of their different behavior in traffic.
Self-driving functions as well need to detect traffic rules indicates by traffic signs or lights. These are very crucial information to further driving functions.
Least the perception needs to identify the environment apart from the street to know what is around. In case of problems, this information is needed to solve unexpected and unknown situations.

All these general groups of information need to be gathered in detail by the perception system. Furthermore, the environment in vehicles limits the systems in terms of computing power and runtime. All these special cases need to be taken into account by designing an appropriate system.

This work will present an in-depth analysis of the current state-of-the-art models and compares their appropriate use in automotive systems. Therefore these models get evaluated in terms of performance, runtime, and classification pattern. The classification pattern returns details on how different situations are handled and if these allow the use in autonomous cars. The object detection is realized by a scene segmentation to return as much information as possible. The lane detection gets trained apart and afterward combined.

1.3 Structure of Work

This work is structured in that the first chapter gives an introduction to the topic and presents the problem this work is dealing with.
The next chapter describes the mathematical and theoretical background to all the

methods and techniques applied during this work. As well it separates this task from other similar techniques.
The following state-of-the-art chapter introduces the best up-to-date investigations in these fields and presents important models.

The main part of this work is the experimental section to build various semantic segmentation models and evaluate them carefully.
The lane detection part builds the second part of the experimental work done in this work. This chapter deals with the problem to detect lane markings and includes these results in the previous ones.

The next chapter brings all results together and the critical discussion about these systems. Here there are also evaluated against the earlier mentioned state-of-the-art models.
The final chapter summarizes the results obtained in this work and comes up with the conclusion of this work.

Chapter 2

Theoretical Background

This chapter describes all the theoretical and mathematical backgrounds to the techniques used for the experimental work done in the following chapters.
First it describes the basics of image classification tasks and focuses on segmentation tasks. Moreover, it explains the importance and uses cases for real-time tasks and models. The last section presents the metrics used to evaluate the results obtained in this work.

2.1 Basics in Image Understanding tasks

Scene understanding tasks with image inputs can be treated very differently. This section introduces the different problems which can be derived from this task. The different problems refer to how the complexity of the scenes is identified. This can contain simple classifications up to more advanced object detection and localization tasks.

The following part with table 2.1 presents the three main tasks for image-based scene understanding and their differences.

Image Classification	Object Detection	Semantic Segmentation
• class level	• instance/class level	• instance/class level
• correct class	• class and position	• pixelwise classification
• output vector	• output structure	• output map

Table 2.1: Comparison of different image based scene understanding tasks

Basically the further left the more details these tasks provide which as well makes the tasks more complex.

The input always is an image which can be defined as an three dimensional (three color spaces) with 256 intensity values like

$$I : \mathcal{U} \rightarrow [0, 255]^3 \tag{2.1}$$

where $\mathcal{U} = [[0; m-1] \times [0; n-1]]$ are the pixel, m and n are the number of rows and columns and 3 is the number of input channels. $I(i)$ is the ith pixel value of the image I, where $i \in \mathcal{U}$.

Like shown in table 2.1, image classification tasks classify the input images in one of a known set of possible classes. Therefor the output is a vector L_C of the number of possible classes C and can be denoted as

$$L_C : \mathcal{V} \rightarrow [0, C] \tag{2.2}$$

The output space $\mathcal{V}$ defines the vector from which one element is chosen to represent the image. This makes the image to be classified as a whole. The disadvantage is that this kind of classification is just applicable to just a few images. Normally an image contains more than just one object. In this case, it can classify the scene shown in the image. But this is still a very limited way of interpreting an image.

Another technique to identify where in the detected object is object localization. It returns the position and size of the detected object. This provides much more information than just classifying an image.
In the case of multiple objects which need to be localized the task is called object detection. This makes object localization a branch of object detection. Object detection can localize multiple and different objects in an image. This includes multiple objects of the same class as well as objects of other classes. The output of an object detection L_D contains two parts $L_{D,C}$ and $L_{D,C}$ as the prediction of the class of the detected object i and its corresponding position.

$$L_{D,i} : \begin{cases} L_{D,C} : \mathcal{V}_C \rightarrow [0, C] \\ L_{D,P} : \mathcal{V}_P \rightarrow [m_i, n_i, w_i, h_i] \end{cases} \qquad \forall \ i \in N \tag{2.3}$$

N is the number of all detected objects and $\mathcal{V}_C$ the classification output space and $\mathcal{V}_P$ the one for its positions. $\mathcal{V}_C$ is the same as for the classification problem and $\mathcal{V}_P$ models its positions where m_i and n_i are the x/y coordinates and w_i and h_i the width and height of the detected object.
This model returns much more information about the image and like that it can be used for many applications. It is like looking for specific objects in an image. This type can classify the detected objects into their classes as well as differences between different objects of the same class. This is called instance detection because it classifies different instances of the same class separately. The problem is that this task detects the object still very vague. It just indicates their position with bounding boxes which are rectangles around the complete object.

The semantic segmentation tackles this problem and classifies the input image pixelwisely. This creates an end-to-end architecture where the output space is an image as well. Every pixel on the output map denotes the class of the specific pixel. This makes a model predict the positions of an object much more detailed and furthermore, it provides information on the specific shape of the detected object of its

corresponding class. Here the label L_S is defined as

$$L_S : \mathcal{V} \to [0, C] \tag{2.4}$$

over $\mathcal{V} = [[0; m-1] \times [0; n-1]]$ where m and n are the same spatial image dimensions as for the input image I. Note that the input space $\mathcal{U}$ and the output space $\mathcal{V}$ equal just in the spatial dimension.
Segmentation problems as well can be broken down onto the instance level. Here it can provide more information as it assigns every object a different id. This does not make sense for all classes. In the case of traffic scenes normally just objects like vehicles or pedestrians are classified on an instance level but not so background classes like trees or buildings. Technically it is possible but not purposeful. This means that the instance segmentation splits a few classes down to the instance level but classifies background classes as a whole.

The characteristic that the output is an image-sized feature map requires a unique type of network for semantic segmentation models. Where classification models mostly are some kind of simple encoder structure segmentation models need some kind of decoder as well. The encoder produces high-level features using convolutions, while the decoder upsamples the feature map to its required shape and helps in interpreting these high-level features using classes. These types of models are called end-to-end models. Figure 2.1 shows the architecture of a basic encoder-decoder network [32].

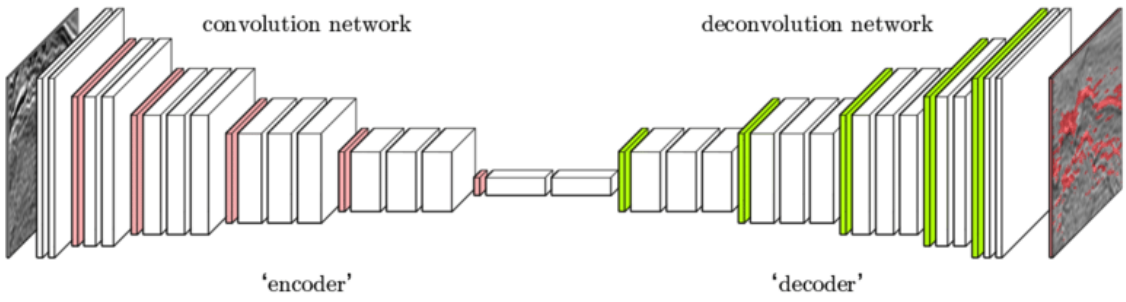


Figure 2.1: Encoder-Decoder Network

The left part describes the encoder which consists normally of convolutional and pooling layers to downsample the spatial size and extend the depth step by step. The red marked layers are the one which downsamples the feature maps and the green ones apply some upsampling techniques.
The decoder network does not have to look exactly like the encoder. There are lots of approaches with smaller decoders and different ways of upsampling.
Basically the models extract all information in the encoder part. The decoder just structures the information into a suitable output map. Segmentation tasks are mostly treated as classification problems so the output map classifies each pixel. Therefore the network defines the classes through the output map.

2.2 Perception Systems in Self Driving Cars

In order to develop intelligent systems for Self Driving Cars (SDC) an extensive perception of the environment has to be done. This is typically made by radar/lidar sensors or camera systems. Because the camera returns much more detailed information about the environment it will be mainly used for future systems. Figure 2.2 [33] presents the basic principle of perception systems used for autonomous or automated driving.

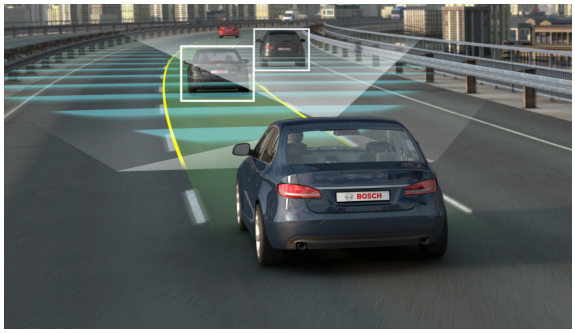


Figure 2.2: Perception system of Bosch with radar and camera

The image presents a combination of radar sensors which are illustrated by the blue waves and two cameras illustrated by the light cones. These are a near range and a common front camera. The near range camera is mainly used for line detection and parking systems and the normal front camera for object detection tasks (also including line detection).

With line and object detection, the image presents two use cases for this perception system. Line detection tasks are used for Lane Departure Warning (LDW) systems that warn the driver before unintentionally leaving the traffic lane. These systems typically use a camera to detect the lane markings on the road in front of the vehicle using computer vision algorithms or neural networks. By detecting the lane markings the driving lane is estimated so that the LDW warns the driver if he passes the line. Another use case is the lane-keeping assistant.

Furthermore the front camera identifies objects in front of the ego car. This enables functions like the Adaptive Cruise Control (ACC) and Forward Collision Warning (FCW). These control the speed depending on the object in front. To do this the exact position, movement, speed, etc. of the vehicle ahead need to be identified. The advantage of the camera is that it can classify the detected objects in very detail what gets very important for example driving in cities. Differentiate between cars, bicycles or pedestrian returns lots of information which can be used for motion control systems. Every object has its own characteristics but detailed classifications enable for example better safety protection systems towards pedestrians or bicycles.

Other functions using object detection are for example traffic sign recognition systems. These typically use the front camera and CNNs for classification. These systems are used to alert the driver if he is driving too fast.

2.3 Scheduling for Real Time Systems

Due to safety reasons and higher predictability autonomous cars use static scheduling for running their tasks. Static scheduling means that every task has its fixed priority and it is predefined when and for how long these tasks will run. Fixed priority scheduling also means that tasks with higher priorities can interrupt tasks of lower priorities. This is called preemptive scheduling. The task then runs in the order defined by the priority in the defined cycle. This cycle is designed depending on the maximum execution time of every task. Current automotive developments support cycles around 5ms, 20ms, and 50ms.

Figure 2.3 shows an example task cycle around the perception tasks in the cycle t. This cycle contains tasks and threads with lower priorities these are called just every n run of the cycle. The first task is the sensor task with threads like reading the camera or sensors.

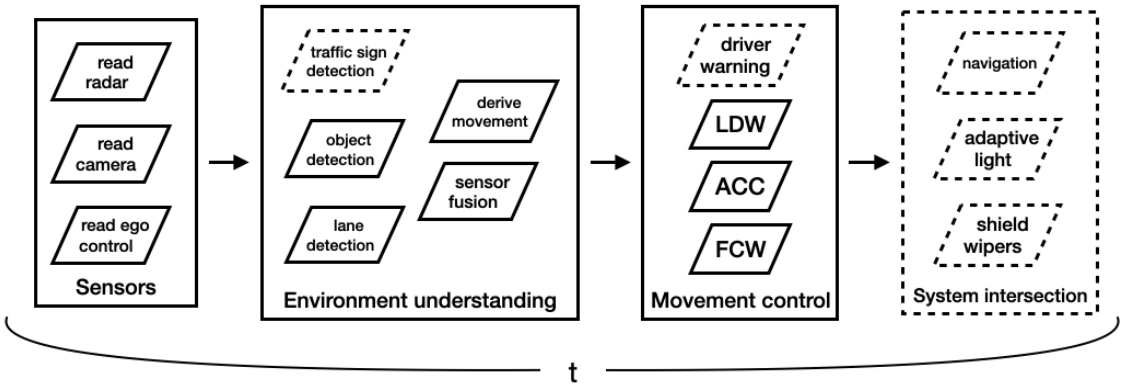


Figure 2.3: Task flow of perception system for autonomous vehicles

This data is loaded by the environment understanding task. Here the perception part takes place with threads like line detection or object detection. The traffic sign recognition is shown in the dashed frame because it will probably run in another cycle because of its lower priority. Just the most important tasks and threads are called in the fastest cycle. After running the detection models the data is fused with the sensor output and the data of the ego vehicle.
This information is then fed into the task where the functions like FCW, LDW or ACC run. These use the generated information and need to run after the environment understanding tasks. This flow needs to run one after another because if the detection models take too long and the radar already returns new values the information in the sensor fusion thread will be mixed up.

Finally some other tasks can run, maybe on another cycle due to their lower priority compared with the essential security and driving tasks.

For real-time systems, a distinction is made between hard and soft real-time systems. Hard real-time systems are designed to always respond in their defined execution time and have to meet their deadlines. Like mentioned before in automotive fields static preemptive scheduling is used. This makes this scheduling predictable and guarantees safety. Every state of the system is predictable and precalculated. This assigns every task a fixed execution time where it has to complete its execution. For embedded systems used in cars, hard real-time systems are used. The following table presents a few important functions and their usual frame rate.

Driving function	frames per second (fps)
Parking function	10
Traffic sign detection	10
Traffic jam pilot	10
Lane keeping assists	50 - 200
emergency braking	50 - 200

Table 2.2: Frame rate requirements depending on different functions

Table 2.2 shows that for basic driving functions a frame rate of 10fps is completely sufficient. Especially parking functions or traffic jam assists operate at very low velocities why no higher frame rate is required.
On the other hand, lane-keeping assists or emergency braking functions are also active on velocities up to 200km/h. The rapidly changing environment makes them require much more data. Also, lane-keeping functions are very sensitive tasks and need smooth control to feel comfortable for the driver. Due to this, the detection tasks need to run much more often.
As well safety regulations are an important point and require more runs of calculating how accidents can be prevented. This leads to clearly higher frame rates.

2.4 Losses and Metrics to assess Segmentation Quality

This section introduces the most important loss functions and error metrics used especially in segmentation tasks.

Loss Functions

This part presents the most important loss functions used for this work. The basic metric for classification problems is the cross entropy loss which is defined as

$$L_{ce} = -\sum_{i}^{C} t_i \log(s_i) \tag{2.5}$$

where s_i denotes the predicted probability of class t_i out of classes C. It just sums up the pixelwise probablistic error for every class. Because of this, this metric is useful for binary as well as for mulitclass classification.
In the binary case when the number of classes $C' = 2$, the cross entropy changes to

$$L_{bce} = -\sum_{i=1}^{C'=2} t_i \log(s_i) = -t_1 \log(s_1) - (1 - t_1) \log(1 - s_1) \tag{2.6}$$

Often it is useful to apply external weights for different classes while calculating the loss. The following equation shows this calculation including the weights w_i

$$L_{wce} = -\sum_{i}^{C} w_i t_i \log(s_i) \tag{2.7}$$

Weighting classes makes the model train more sensitive on different classes. This can help to compensate bigger imbalance in the dataset or just put a bigger focus on certain classes.

Metrics

The confusion matrix is an error matrix used to evaluate statistical classification problems. It provides a detailed visualization of the predictions made by any model compared with their labels.
The correct labels are listed as the rows of the table and the predictions over the columns. This creates a structure where the correct classifications are located on the main diagonal. The big advantage of this graphic is that it shows which errors the system is making. The elements of the confusion matrix are called True Positives (TP) for the correct classification of positive values, False Negatives (FN) for the false classification of positive labels, False Positives (FP) for the false classification of negative values and True Negatives (TN) for the correct classification of negative values. This creates the typical representation which is shown in table 2.3.
As well as for the binary case the confusion matrix can be computed for multi-class problems. The positive respective negative classifications are now represented by the name of the class. The correct classifications are still located on the main diagonal of the matrix.
This presentation and all the information enable the usage of more adequate metrics than the accuracy. The accuracy measures the percentage of how many pixels are

		Prediction	
		positive	negative
Label	positive	TP	FN
	negative	FP	TN

Table 2.3: Confusion Matrix

		Predictions			
		Class 0	Class 1	...	Class n
Labels	Class 0	true	false	...	false
	Class 1	false	true	...	false
		⋮	⋮	⋱	⋮
	Class n	false	false	...	true

Table 2.4: Multi-Class Confusion Matrix

classified correctly. This is a very basic evaluation that misses important information regarding the different classes.

The recall is a metric that returns the performance on every true class separately. This is effective for small classes which tend to hardly be classified correctly. It is defined as

$$Recall = \frac{TP}{TP + FN} \tag{2.8}$$

To measure the correct classifications out of all positive predictions the precision metric is used. This indicates if the model suffers from lots of False Positivess.

$$Precision = \frac{TP}{TP + FP} \tag{2.9}$$

Models with a high recall can still have a low precision or vice versa. Therefore the F1-Score represents the combination of these two metrics. It is defined as

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{2.10}$$

This score combines both metrics using a harmonic instead of arithmetic mean by stronger pushing the extreme values.

Another metric based on these logical groups is the Intersection over Union (IoU). This is a common evaluation metric for object detection tasks where it evaluates the performance of the localization sub-problem. This makes this metric also very useful for segmentation tasks. It is also often called the Jaccard index and uses the different logical groups calculated by a confusion matrix. The IoU is defined as

$$IoU = \frac{| p \cap gt |}{| p \cup gt |} = \frac{| TP |}{| TP + FP + FN |} \tag{2.11}$$

where p is the prediction and gt the ground truth label which are areas. The IoU is the overlap of these two areas.
Normally the mean IoU is used to describe the performance of a model. Here the IoU is computed over all classes and then averaged

$$mean\ IoU = \frac{1}{N} \sum_{i=1}^{N} IoU_i \tag{2.12}$$

Here the influence of every class is equal. This means that bigger classes do not have a bigger influence on the performance.

Chapter 3

State of the Art

This chapter presents the relevant developments in the respective fields of investigation. It first presents the published literature for semantic segmentation models and then the currently leading models in lane recognition.

3.1 State of the Art in Semantic Segmentation in Real Time

Semantic Segmentation is a quite new approach to analyze and classify images. It uses a pixel-wise end-to-end classification to classify every pixel individually.
To do this, there are presented several approaches from the last few years but not everyone is suited for semantic segmentation in real-time. Some are very expensive in terms of calculation time and memory size. This makes them not suited for real-time systems used for autonomous driving but still provides good investigations.

The very first known approach for semantic segmentation is made by Long et al. [12] who proposed a Fully Convolutional Network (FCN) adjusted to semantic segmentation. They adapted leading classification networks like AlexNet [**?**] or VGG [**?**] into fully convolutional networks and apply fine-tuning to adjust the network to the tasks of semantic segmentation. In general, this model uses a usual contracting neural network and instead of classifying in the last step, it keeps the spatial size of the coarse outputs and upsamples them again. Therefore the final classification layers are removed and the Fully Connected (FC) layers are converted to convolutional layers. Through 1x1 convolutions and deconvolutions, the coarse output is upsampled to the original size for pixel-wise predictions. As well skip connections are used to recover spatial details from encoding structure.
The good results on the PASCAL VOC 2011 dataset [**?**] (mean Intersection over Union (IoU) of 62.7) have proven the advantages of FCN for semantic segmentation.

Based on this approach Ronneberger et al. [13] developed the U-net which surpasses the performance of the FCN. This approach mirrors the contracting part and

applies successive upsampling to obtain the same coarse outputs. Therefore it replaces the MaxPooling layer with upsample layers. So the whole network results in a U-structure. They found that concatenating output maps of the same size increases the performance. Also, the successive upsampling increases the resolution of the coarse outputs, which assembles to effectively combine high-level features with low lever features. Every time the spatial size of the outputs is reduced, the depth is doubled to propagate context information through the network.

The SegNet of Badrinarayanan et al. [21] is another architecture derived from the FCN and the U-net. It proves that successively upsampling of the output map increases accuracy. More importantly, it introduced a connection of the max-pooling indices received from the corresponding encoder to perform non-linear upsampling of the decoder input feature maps. The efficiency of this network comes from its extensive reduction of layers.

Another improvement of this encoder-decoder structure is presented by Paszke et al. as the ENet [14]. Instead of just stacking convolutional and pooling layers onto another in the encoder structure, they introduced the effect of residual bottleneck blocks. ResNets have proven their big effects on image classification and enable building bigger networks more efficiently. Fast downsampling in the initial layers reduces drastically the network size and makes the filters extract more context. On the other hand, by downsampling the images, important spatial information is lost. This downsampling is done by concatenating a 3x3 convolution and a 3x3 max pooling. More context can also be gathered by the use of dilated convolutions, which increases the receptive field, but does not increase the size of the output maps. Here it is shown that the use of big decoders is quite small and the network focuses on the encoding (like in classification tasks). The decoding part is just used for upsampling and fine-tuning the details. To not have to downsample the feature maps any further and still gather more context information dilated convolutions (various dilation ratios) at the smallest feature map size are applied. These results are obtained by alternating dilated and normal bottleneck blocks.
Evaluated on the, for autonomous driving relevant cityscapes dataset, an accuracy of 58.3 with real-time performance of 21.6 frames per second (fps) on an input image size of 1920x1080 on an NVIDIA Titan X GPU is achieved. Compared to usual ResNets this network sacrificed the accuracy by removing layers to gain efficiency.

The DeepLab from Chen et al. reaches state-of-the-art performance by extensively using dilated convolutions to control the resolution at which feature responses are computed. For better localization of segments of little feature maps a CRF is applied, which highly increases the network's computational overload. Although the dilation increases the performance, it is due to their larger receptive fields computationally more expensive. Moreover, they introduce the advantages of pyramid pooling to be more robust against objects at different scales.
Despite its new introductions and good performance this architecture is not suitable

for real-time systems.

Zhao et al. proposed in the Pyramid Scene Parsing (PSP) Network [16] the Pyramid Pooling Module (PPM) which turns out to increase the accuracy because it combines features at different levels. These different pooling outputs are concatenated to build the final feature map. Due to the PPM it is capable of connecting the global information of different regions based on context aggregation.

Zhao et al. propose an Image Cascade Network (ICNet) that processes the input at different resolution scales at different branches. These different branches get aggregated by the cascade feature fusion which enables high-quality segmentation. Basically in this module the feature maps get upsampled to the same size, dilated convolutions are applied and the feature maps are summed up.

The encoder-decoder structure for semantic segmentation was further developed by Romero et al. who proposed the ConvNet architecture [7]. Due to the big advantages to avoid the degradation of ResNets, also for semantic segmentation the residual structure plays a big role in ConvNets. Because deeper networks are too inefficient to work as real-time systems the type of residual blocks to non-bottleneck blocks are changed. Although the bottleneck version is known to be more efficient because it reduces the computed feature maps to reduce computation time, there is no big difference for shallow networks like the here presented architecture due to their increased feature dimensions. Non-bottleneck blocks are made up of stacked 1x3, 3x1, 1x3, and 3x1 convolutions and the addition of the shortcut. This results in faster execution, a reduced number of parameters, and better regularization. It is claimed that through this bigger width, predictions can be improved at the cost of efficiency. At the smallest output size again dilated convolutions are used to allow exponential expansion of the receptive field without loss of resolution or coverage. Like the ENet, this structure downsamples the input image right at the beginning just a few times to not sacrifice too much pixel accuracy.

Mazzini et al. proposed a Guided Upsamplin Module (GUN) [17] which introduces a new way to upsample feature maps efficiently. It is based on two branches with input images at different spatial sizes and with shared weights to extract fine and coarse features. Before decoding, these branches get fused and the novel guided upsampling module is applied. This module takes into account the high-resolution features from the earliest layers with bigger resolutions as well as complex features from the deepest ones. Due to the efficiency of this module, this network is also recognized as a real-time system with state-of-the-art accuracy an IoU of 70.4.

The presented approaches are all dealing with the trade-off between downsampling the image to capture more context information at low calculation cost and better pixel accuracy through more detailed information with high-resolution maps.
Poudel et al. defined with the ContextNet [8] a different approach to the com-

mon encoder-decoder architecture. They defined a two-branch network that focuses on both parts separately and adds them together before the decoding starts. As demonstrated in [22], the combination of context information from maps of different resolutions improves the performance. So for the high-resolution branch according to runtime speed, a shallow network is applied to capture detailed spatial features. Whereas the low-resolution branch is deep enough, to capture semantically rich features.
In detail, the shallow network applies directly depth-wise separable convolutions on the original input which reduces the execution time a lot due to fewer parameters and floating-point operations. Depth-wise separable convolutions are introduced by MobileNet [18] and describe a 1x1 pointwise convolution applied directly after a depth-wise convolution.
The low-resolution sub-network applies various bottleneck residual blocks on spatial decreased input images. These bottleneck blocks apply two 1x1 convolutions with a depth-wise separable convolution in between. The first convolution expands the number of channels by a predefined factor to run the depth-wise separable convolution on the bigger feature map. The last convolution restores the output depth. This extension of the feature map makes the network learn more complex features and increases the accuracy.
This work presents an effective model to combine global and local context to archive leading results in real-time segmentation. On the cityscapes dataset, a mean IoU of 66.1 is obtained with 18.3 fps on an NVIDIA Titan X.

Based on the introduction of two branch models and skip connections of general encoder-decoder structures Poudel came up with the Fast-Segmentation Convolutional Neural Network (SCNN) [9]. Just like in encoder-decoder structures there are applied a few layers to downsample the input to extract low-level features and then split it into two branches. This model sticks to the efficiency of depth-wise separable convolutions which are used in the initial layers as well as in the global feature extracting branch. It applies various bottleneck blocks which also apply the depth extension inside these blocks. Successively the feature map depth gets expanded between the bottleneck blocks. At the end of this branch, it is stated that a PPM increase performance by aggregating different region-based context information.
The other branch gets not further processed for recovering the spatial details. In consequence of efficiency, both branches are summed up and apply just three more layers to upsample the output map.
The Fast-SCNN presents an above real-time application that outperforms every other one in fields of runtime. Furthermore, the mean IoU of 69.15, is very high compared to other real-time networks.

Another different structure which is based on encoder-decoder is the recently presented ShelfNet [11] by Zhuang. This structure is described by a stacked encoder-decoder with multiple skip connections. The first part of it is a very shallow backbone network like the ResNet18 which reduces the spatial size several times. Now

there is applied a decoder branch with a residual convolutional block and an upsampling layer at every level of the backbone network (for every spatial size). This structure is applied two times more to generate a downsampling encoder and an upsampling decoder. Skip connections feed the output maps of the last block of the same size into the residual blocks. The residual blocks consist of two convolutional layers which are sharing their weights what reduces the parameters drastically.
As a backbone network, there can be used different CNN architectures to compute the different levels of output maps.
Due to the many skip connections and residual connections the ShelfNet provides multiple paths of information flow. Like a ResNet it ensembles information from a variety of deep and shallow levels. The publishers found that increasing the number of information paths instead of increasing parameters can improve accuracy.
Dilated convolutions on a little spatial size with increasing dilation ratios have the same effect as stacking various encoder-decoders together but it is limited to the information in the feature maps.
The results (IoU of 74.8) obtained by this model surpass every other real-time segmentation model.

In 2016 Iandola et al [23] presented with the SqueezeNet another very efficient approach to object detection through the introduction of a brand new fire module. This paper mainly threatens the goal of reducing parameters by obtaining the same level of accuracy. This is made by the fire module which makes extended use of 1x1 convolutions instead of 3x3 convolutions and of reducing the size of input channels fed into a bigger 3x3 convolution. The mentioned fire module contains a squeezing and an expanding part. It is constructed by a 1x1 convolution to reduce the depth of the feature maps and followed by two expanding convolutions (1x1 and 3x3 convolution) which get concatenated to form the output feature map.
Their architecture contains 9 fire modules which increase the depth and reduce the spatial size of the feature maps. Furthermore, it is stated that delayed downsampling of the model architecture leads to better classification accuracy.
Nanfack et al adjusted this approach to the segmentation task by creating the Squeeze-SegNet [24]. The downsampling part stays completely the same as in the original SqueezeNet architecture and they just add an upsampling part. Therefore they simply create an u-net-like architecture where reverse fire modules are applied to upsample the feature maps. This reverse fire module is designed as a series of concatenations of expanding modules followed by a squeezing module.
Here it is difficult to evaluate their results because this architecture is not tested on the cityscapes dataset why the results cannot be used as a reference. Furthermore, as metrics, they used the accuracy instead of the mainly used mean IoU for segmentation tasks. Due to the extremely efficient model and its fewer parameters this architecture is also considered as state of the art for this work.

3.2 State of the Art in Lane Recognition

The task of lane recognition is a specific task in the fields of semantic segmentation. This task performs a semantic segmentation with the two objects - lines and background. Sometimes lines get separated to correctly identify which line (traffic lane marking, centerline marking, road limiting line) it is.
This means for this task there can be used every model suited for general segmentation tasks too.
In the following, there are presented some important papers which highlight different well-proven architectures and approaches.

First line detection approaches are mainly based on preprocessing steps like applying filters to reduce mean, noise or Finite Impulse Response (FIR) filters, detection of ROI, thresholded segmentation/binarization or color space transformations. The feature extraction part was realized through edge filters and hough transformations to predict the lines in the image.

Through the introduction and great success of Convolutional Neural Network (CNN)s, these were also applied to the task of lane detection. In 2014, J Kim and M Lee [26] proposed a CNN consisting of multiple convolutional layers for encoding and some fully connected layers to generate the output map. The FC layers return the list of output nodes (lane or no lane) which are then reshaped to generate the final output map. This model also requires preprocessing steps like ROI selection as well as edge detection filters.
Next architectures based on object detection tasks transfer these models to the task of line detection. The approach of the Overfeat approach is to predict the lines by returning the starting and endpoints a vector [27]. The idea is to estimate the structural or geometric information by the CNN like for object detection tasks.

The introduction of end-to-end architectures like segmentation models enabled many new approaches. Applying deep CNNs to semantic segmentation to detect lines can be difficult because of the narrow sizes of the lines. For example, excessive pooling layers can delete lots of information. By 5 2x2 max-pooling layers the line needs to contain more than 32 pixels to be reconstructed. Due to that, it is important to use learnable layers to shrink the spatial sizes. Dilated convolutions help to gather a bigger spatial context without the need for pooling layers but are very expensive in terms of real-time performance.
In 2018 Shao-Yuan Lo et al. proposed Lane Marking Detection (LMD) based on the VGG architecture. They implemented dilated convolutions between the encoder and decoder parts. The decoder outputs the original-sized feature map indicating the lines as class 1 and does not require any preprocessing steps.

While driving the lanes change continuously and linearly. By combining multiple frames the information of previous frames can be used to add the temporal

correlations to the spatial or geometric features. This prior knowledge can boost performance as shown by Qin et al. in 2019 by proposing a CNN-LSTM combination network [29]. The temporal information is fused between the encoder and decoder where 2 LSTM layers are added. This LSTM runs on a sequence of 5 encoded frames and returns the same number of frames to the decoder. Although its benefits this algorithm is resource consuming and less efficient.

The Spatial CNN proposed by X. Pan et al. introduced in 2017 a new approach to increase performance by integrating a new convolutional module for the high-level features which is applied after the encoding CNN. As well they stated the benefits of using multiple loss metrics. Besides the spatial cross-entropy loss for pixel-wise classification, they also use for every class a sigmoid classification loss which indicates if this specific line appears in the image. This enables a more stable and accurate prediction model for detecting various lines.

Due to the most precise predictions the end-to-end architectures turned out to work best for line segmentation tasks, as well as for other image segmentation problems. Especially by dropping the need for pre- and post-processing, the image makes this architecture that powerful. Object detection algorithms always need to create lines out of the detected markings.

Chapter 4

Experimental Work - Semantic Segmentation

This chapter describes the experimental part done in this work. The main part is the development and evaluation of various models to reproduce and exceed state-of-the-art performances.
The first section presents the data sets used in this chapter and explains how these are used in this work.
The next section presents the different models implemented and developed, their advantages, and the potential to surpass its performance.
Next, there are shown various approaches to control and improve the training of the different models like using data augmentation techniques, clustering of classes for classification pretraining the model on different data, or adjusting the resolution size of the input images.
Finally the results of this work are shown and compared among all models. Also, the results are evaluated with the state of the art references to evaluate the overall performance.

4.1 Data Set

This section presents the data sets for semantic segmentation used for this work. These are the dataset cityscapes and mapillary vistas.

4.1.1 Cityscapes

The main dataset, on which the model is built and trained is the cityscapes dataset [19] published by Daimler and an association of German universities. It contains large-scale images and videos annotated on pixel or instance level for semantic labeling. Moreover, the publisher provides coarse annotated labels for additional data. This works exclusively with the pixel-level semantic labels and without the coarse annotated data.
In general the data contains 5000 images from 50 different German cities and are

split into sets for training with 2975 samples, for validation with 500 samples, and testing 1525 samples. For the testing set, only the images are open to the public. The different sets split up the images by cities. All images have originally the size of 1024x2048. The dataset contains 34 classes of relevant objects for autonomous driving.
This dataset is also the most common dataset to compare results to the state of the art.

For training and evaluation there are just used 19 of these classes and the other ones are joint to a class of other objects. This generalization of its classes is used to focus on objects which are relevant for the task of autonomous driving and to filter small classes with lower occurrences in the dataset. The way of joining the classes into bigger ones is shown in table 4.2 on page 26.
Here the first column shows the real object id which classifies the objects into the 34 specific classes. The second column called Id_{train} describes the 19 more general classes. Here all objects which do not belong to one of the important classes are assigned the Id 19. The important classes are selected by attributes such as importance for autonomous driving and frequency of occurrence of its objects in the dataset. The third column just keeps the name of the class.
For now the fourth column is not relevant and just gets relevant when this dataset will be combined with another one.

This dataset can also be evaluated because of its object groups. Therefore some classes are usefully grouped for evaluation (not for training). This helps to better analyze the behavior of the model. Some classes are closer to some classes than to others. E.g. the classes person and rider are very similar or bicycle and motorcycle or truck, bus, caravan, and trailer. The groups are defined by the publisher of this dataset and are shown in table 4.1.

Flat	Construction	Object	Nature	Sky	Human	Vehicle
road	building	pole	vegetation	sky	person	car
sidewalk	wall	traffic light	terrain		rider	truck
	fence	traffic sign				bus
						caravan
						trailer
						train
						motorcycle
						bicycle

Table 4.1: Groups to evaluate category accuracy

This table shows the logical groups of classes for better evaluation. Every but the last group contains two to three classes. The vehicle group includes all vehicles, whether two-wheeled or four-wheeled. Not listed is the group which contains all

the other classes which are labeled with 19 during training. These are lots of little classes which cannot be defined very well because of their big variance.

4.1.2 Mapillary Vistas

Another very important dataset in this work is the mapillary vistas dataset [20]. It is still not that common as the cityscapes set yet, but it has its advantages. It contains 25000 large-scale street-level images so it is five times larger than the cityscapes set. Furthermore, it classifies the objects in 66 instead of 34 semantic classes. Moreover, the images are taken from all continents which makes it way more complex and increases its variance. Where cityscapes just contain different German cities the mapillary includes more information. On the other hand, it got a stronger diversity of the weather conditions containing rain, fog, snow, and changing daylight. This richness of details extends the information significantly compared to all other data sets. All these elements make this dataset more realistic and should be used for training autonomous driving tasks.
Due to its less spreading it is mainly used for pretraining because the results cannot be compared to other models.

4.1.3 Converting mapillary Vistas labels to labels of cityscapes

The mapillary vistas dataset will be used to pretrain the models for the main training on the cityscapes dataset. As described in section 4.1.2 the images of the mapillary dataset are classified into 66 classes. To convert them into the 34 respectively 19 classes, the assignments shown in table 4.2 on page 26 are used.
The previous section already introduced the first part of the table which describes the ids of the labels of the cityscapes dataset. Now the 66 classes of the mapillary dataset are allocated into these classes. Not just into the 34 detailed classes but also into the 19 classes to train on.
This assignment enables a way to pretrain on this more complex and highly different dataset.

4.2 Presentation of used models

This section presents the used models and their developments applied to them. These are strongly connected to the ones presented in the section **??**.
The models focused on are the Fast-SCNN, the ShelfNet, the Squeeze-Seg-Net, and the ConvNet.

4.2.1 Fast-SCNN

This section presents the Fast-Segmentation Convolutional Neural Network (SCNN) which is an advancement of a typical two branch model. Multiple branches which can also be seen as skip connections are used to connect global context with detailed

Id	Id_{train}	Label Cityscapes	Label Mapillary Vistas
0	19	unlabeled	unlabeled
1	19	ego vehicle	ego vehicle
4	19	static	mountain, bench, billboard, fire hydrant, junction box, mailbox, phone booth, trash can, car mount, banner, cctv camera
5	19	dynamic	ground animal, bird, wheeled slow, boat, other vehicle
6	19	ground	water, pedestrian area
7	0	road	bike lane, service lane, crosswalk (plain), road, lane marking (crosswalk), catch basin, lane marking (general), manhole, pothole
8	1	sidewalk	curb, curb cut, sidewalk
9	19	parking	parking
10	19	rail track	rail track
11	2	building	building
12	3	wall	wall
13	4	fence	fence, bike rack
14	19	guard rail	guard rail, barrier
15	19	bridge	bridge
16	19	tunnel	tunnel
17	5	pole	pole, traffic sign frame, traffic sign (back), utility pole, street light
18	19	pole group	
19	6	traffic light	traffic light
20	7	traffic sign	traffic sign (front)
21	8	vegetation	vegetation
22	9	terrain	terrain, sand, snow
23	10	sky	sky
24	11	person	person
25	12	rider	bicyclist, motorcyclist, other rider
26	13	car	car
27	14	truck	truck
28	15	bus	bus
29	19	caravan	caravan
30	19	trailer	trailer
31	16	train	train
32	17	motorcycle	motorcycle
33	18	bicycle	bicycle

Table 4.2: Transfer mapillary labels into cityscapes labels

boundaries. Therefore the shallow part learns the boundary information on a bigger resolution size and the deeper branch learns the global context on a much smaller resolution.
This structure of the Fast-SCNN network which can be divided into four parts is graphically shown in figure 4.1. The network is a general encoder-decoder structure with one skip connection after the first three convolutions. Furthermore, it makes use of depthwise separable convolutions to reduce runtime.

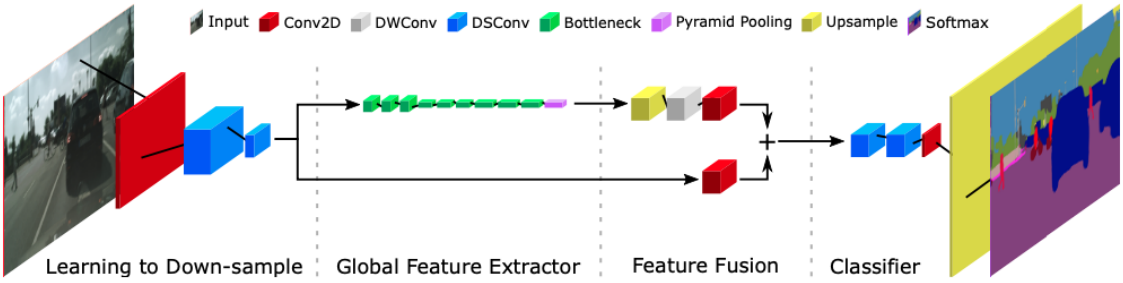


Figure 4.1: Fast SCNN network with explication of different layers

The first part describes the downsampling using normal convolutions and separable convolutional operations. Due to efficiency reasons the network mainly uses separable convolutions. However, because of the just three input channels of the first map, there is applied a basic convolutional filter to not waste important information and to extend the number of channels very early in the network. The first three convolutions are used to downsample early in the network to reduce spatial size while its depth increases.

The next part is called the *Global Feature Extractor* part and uses a specific bottleneck structure called *bottleneck residual block* introduced by MobileNet-V2 [18]. These expand the feature channels inside this structure to apply the main convolution on a bigger tensor. Due to efficiency, this structure uses depthwise convolutions as convolutions with bigger kernel sizes. This bottleneck structure is shown in table 4.3.

Input	Operator	Kernel	stride	Activation	Output
h x w x c	Conv2D	1x1	1	ReLu	h x w x tc
h x w x tc	DWConv	3x3	s	ReLu	$\frac{h}{s}$ x $\frac{w}{s}$ x tc
$\frac{h}{s}$ x $\frac{w}{s}$ x tc	Conv2D	1x1	1	-	$\frac{h}{s}$ x $\frac{w}{s}$ x c'

Table 4.3: Specific bottleneck structure of Fast SCNN

Here h, w and c represents the height, width and the channel number of the feature map, t the expansion factor which temporarily expands the feature map and s defines the stride for the corresponding convolution.

This means the first Convolution extends the channel number of the feature map by an extension factor with an efficient 1x1 Convolution. Then the depthwise convolution with the bigger kernel size of 3x3 gets applied on the extended feature map to capture more information at once.
The effect of the depthwise convolution was also empirically proven by MobileNet-V2. They state that the accuracy just drops a very little bit but with around 8 times smaller computational cost. The computational cost of a depthwise convolution is shown in calculation 4.1.

$$cost = h_i \cdot w_i \cdot d_i \left(k^2 + d_j\right) \tag{4.1}$$

where h_i is the height, w_i the width and d_i the depth of feature map i, k is the kernel of the applied convolution and d_j the output depth.
In comparison a normal convolution multiplies also the output depth and results in much higher computationally costs.

$$cost = h_i \cdot w_i \cdot d_i \cdot d_j \cdot k \cdot k \tag{4.2}$$

The last operation in the bottleneck structure is a normal convolution to restore the output depth c' of the feature map.
This bottleneck block gets applied three times to gather global features on the small spatial size. Here the network downsamples a lot slower by applying the same block several times. Table 4.4 describes the exact parameters used in the different layers. t defines the expansion factor used for the bottleneck blocks, c the depth, n the number of repetitions, and s the stride of the convolutions.

Input	Block	t	c	n	s
256 x 512 x 3	Conv2D	-	32	1	2
128 x 256 x 32	DSConv	-	48	1	2
64 x 128 x 48	DSConv	-	64	1	2
32 x 64 x 64	Bottleneck	6	64	3	2
16 x 32 x 64	Bottleneck	6	96	3	2
8 x 16 x 96	Bottleneck	6	128	3	1
8 x 16 x 128	PPM	-	128	-	-
8 x 16 x 128	FFM	-	128	-	-
32 x 64 x 128	DSConv	-	128	2	1
32 x 64 x 128	Conv2D	-	19	1	1

Table 4.4: Customized Fast SCNN network with slower upsampling part

The next part is called the *Feature Fusion Module* because it connects the two branches. One branch is a skip connection from the early downsampling part and

the other one is the output feature map of the bottleneck structure. On the skip connection, the Feature Fusion Module (FFM) applies just a 1x1 convolution while on the other branch it needs to apply an upsampling layer. Furthermore, it applies a 3x3 depthwise convolution and a 1x1 convolution.
Due to efficiency reasons the two branches just get added instead of fusing them more sophisticated. This addition is followed by a ReLU activation.

The last part, the upsampling in figure 4.1 is called the *Classifier*. Here two efficient depthwise convolutions and a normal convolution are applied to create an output that spatially matches the input and consists of the expected 19 output classes. As shown in figure 4.1 and specified in table 4.4 it applies a fast upsampling by just upsampling and a few convolutional layers.

Results Fast SCNN

The model is trained on the cross-entropy metric and for evaluation, the pixel accuracy and the mean Intersection over Union (IoU) are used. The mean IoU is the most common and representative metric for image segmentation tasks. The results are shown in table 4.5.

Metric	Result
Pixel accuracy	0.9055
Mean Intersection over Union	66.42
Runtime [s]	0.2524

Table 4.5: Segmentation results of Fast SCNN

The accuracy used to be much higher than the mean IoU. The IoU is a very critical measure for segmentation tasks because it averages the performance overall classes. So small classes can have a bigger influence on the results which makes this metric way more representative than the accuracy. State-of-the-art results for segmentation on the cityscapes dataset reach values between 65% and 70%.

This network is trained with the adam optimizer and with batches of 4. Due to the big image resolution, this is the limit of the available machines. The batches are generated by a data generator applying different data augmentation techniques on random examples of the shuffled dataset. For data augmentation, random flipping, resizing between 0.5 to 2, translation of up to 20%, channel shifts up to 20, and brightness noise from 0.5 to 2 are applied. These techniques are justified and explained later on in section 4.3.2 Compare data augmentation techniques.

Figure 4.2 shows the evolution of accuracy and loss during training. The left figure shows the accuracy while the right one shows the loss of the training over 1000 epochs.

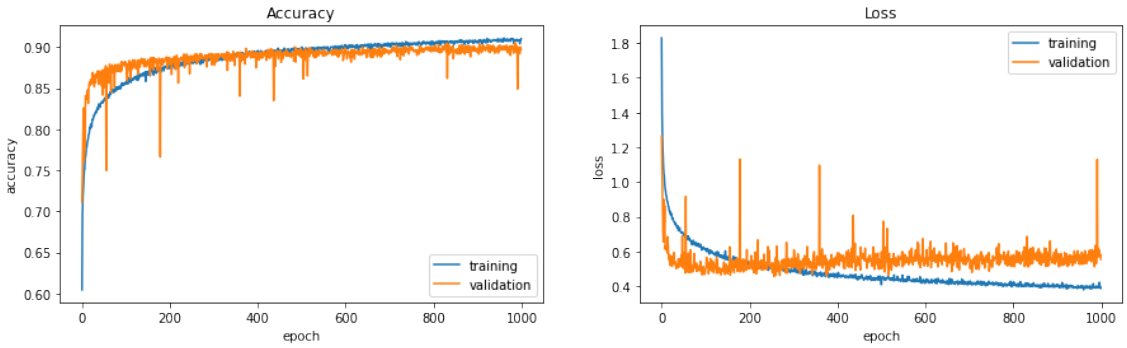


Figure 4.2: Accuracy and loss of training of Fast SCNN over 1000 epochs

First thing to observe is that the model is trained very well but still tends to slightly overfit because the validation loss is raising while the loss on the training data still decreases. However the validation accuracy still increases, so there is no need to stop the training earlier. Just after 1000 epochs the accuracy stagnates and does not increase any further. For semantic segmentation tasks, this is common training progress. The good tuning of this model also shows the fact that the accuracy of training data (0.9102) and validation data (0.9055) are very close.

4.2.2 ShelfNet

The ShelfNet is a new network structure that is strongly based on the principle of encoder-decoder networks. As shown in figure 4.3 it is separated into two main parts - the backbone network and the segmentation shelf.
As backbone network, every encoder structure can be used. Here skip connections feed the feature maps of different spatial sizes into the segmentation shelf. The segmentation shelf adds multiple encoder-decoder structures to the model to add more depth to the network. Because of the better information flow through a bigger number of paths created by the skip connections, segmentation accuracy can be improved with just a few runtime increases.
Figure 4.3 shows the shelfnet structure divided into two parts. As a backbone network there can be used every encoding CNN which reduces the spatial size of input tensors at least four times. In this network a ResNet18 is used which downsamples the feature maps on levels A to D to spatial sizes of $\frac{1}{4}$, $\frac{1}{8}$, $\frac{1}{16}$ and $\frac{1}{32}$ with depths of 64, 128, 256 and 512 respectively. The backbone network serves as the first encoding step applied to the network.
Due to efficiency reasons these feature maps are reduced in depth by the green blocks when introducing them into the segmentation shelf.
Following calculation of the computational cost show the effect of this depth reduction. H and W are spatial sizes, K is the kernel size, and C_{in} are C_{out} are the channel number of input and output respectively.

$$cost = H \cdot W \cdot K^2 \cdot C_{in} \cdot C_{out} \tag{4.3}$$

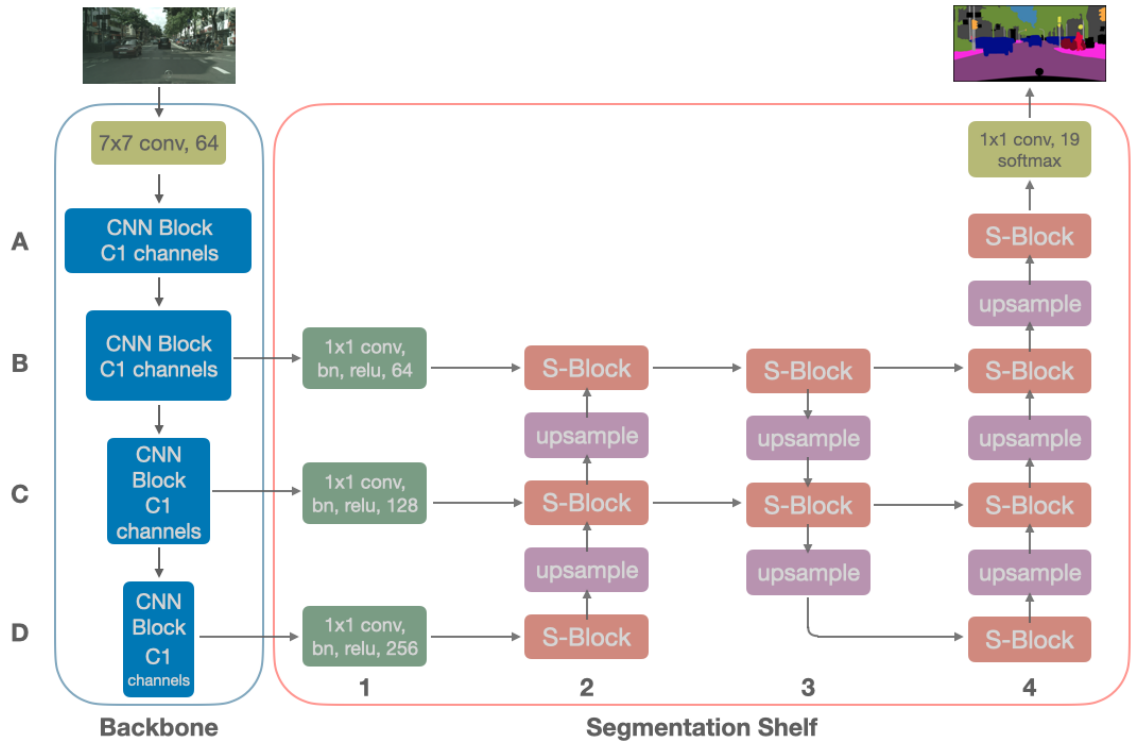


Figure 4.3: Network structure of ShelfNet which introduces the segmentation shelf as multiple encoder-decoder pairs

The reduction of the input and output channels to its half reduces the computational cost quadratically. With this, it is possible to add more encoder-decoder pairs to the segmentation shelf without surpassing the original computation cost.
The depth created in each block defines the depth levels for the complete segmentation shelf. The green blocks are formed by a sequence of a 1x1 convolution, batch normalization, and a ReLU activation.
Here the segmentation shelf applies in column two the first decoder structure by continuously upsampling the smallest feature map. Therefore the shelfnet uses specific structures called s-blocks. Before every s-block in the segmentation shelf there are added skip connections (shown in figure 4.3) before applying the convolutional operations in the s-block.
Figure 4.4 described all the operations applied inside the s-block. First, there are added the skip connections of the previous column (encoder/decoder), then two 3x3 convolutions with shared weights are applied. Both are followed by batch normalization layers and the first one got an ReLU activation. Between both convolutions, a dropout layer is applied to avoid too fast overfitting inside this block. At least the residual shortcut gets added and a ReLU activation is applied on the output.
To make the network suitable for real-time applications level A of the segmentation shelf is skipped and the second encoder starts again at level B in column 3. The process inside the segmentation shelf stays the same.
In column 4 the decoder upsamples the output feature map up to the final resolu-

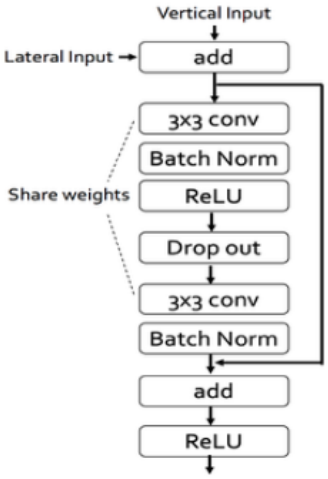


Figure 4.4: S-Block: Residual block with shared weights used in segmentation shelf

tion. A last 1x1 convolution creates the 19 output classes with a softmax activation.

The skip connections between the encoder and decoders generate lots of different paths for information flow which increases segmentation accuracy. These different paths create an ensemble of different FCN networks. Andreas et al. [**?**] has proven that ResNets because of their residual connections improve performance because they work as an ensemble of different FCNs. This principle got archived in the ShelfNet by its skip connections.

Results ShelfNet

This model is also trained on the cross-entropy metric and for evaluation, the pixel accuracy and the mean IoU are used. The following table shows the results in categories of accuracy, mean IoU, and runtime.

Metric	Result
Pixel accuracy	0.9390
Mean IoU	73.47
Runtime [s]	0.3123

Table 4.6: Segmentation results if ShelfNet

The performance results in the table show that this network exceeds the results of the earlier presented model in terms of accuracy as well as mean IoU. The accuracy is just a little higher but especially the mean IoU metric shows with 73.47% the big difference in performance.
This good performance comes in the cost of runtime like it is shown in table 4.6. This network got with 13.420.063 parameters, more than 7 times more parameters than the efficient fastscnn. But that many parameters do not harm runtime that

much due to the effective channel reduction and by creating a shallow segmentation shelf. The different paths for information flow in the segmentation shelf lead to high accuracy.

Figure 4.5 shows the training over 1000 epochs where the best model is saved. On the left, it shows the progress of accuracy, and on the right the loss value.

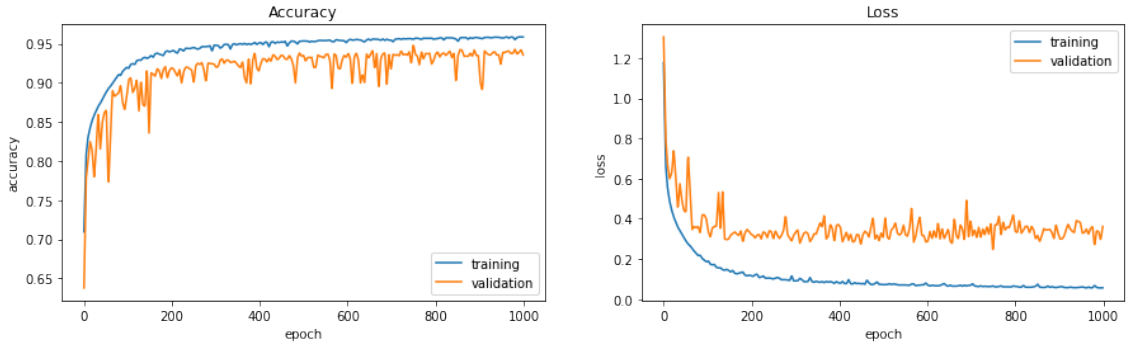


Figure 4.5: Accuracy and loss of training with ShelfNet over 1000 epochs

The learning progress shows that the model is designed very well because the performance on training and validation data is close and the model is not overfitting. The accuracy on validation data is a little lower than on the training set during the whole training and reaches just after 400 epochs a level of saturation where the performance just improves very little. From there on the loss value does stay nearly the same but keeps oscillating.

4.2.3 ConvNet

The ConvNet describes a typical encoder-decoder network (presented in figure 4.6) which bases on redesigned bottleneck blocks including residual connections. Unlike many ResNets which add more depth to their network to achieve better performance, this model proposes a wider architecture that makes extremely efficient use of its minimized amount of layers.
To generate this width the ConvNet uses a different bottleneck structure called *non bottleneck 1D* explained in figure 4.7a. Key factors are the convolutions with factorized 1D kernels which lead to fewer parameters and faster execution. These consist of four stacked one-dimensional convolutions with a specified width w with a residual connection. As shown by calculation 4.4, these convolutions (left) just have $\frac{2}{3}$ of the parameters that a normal bottleneck block (right) consisting of two 3x3 convolutions would have.

$$H \cdot W \cdot 2 \cdot K \cdot 1 \cdot C_{in} \cdot C_{out} < H \cdot W \cdot K^2 \cdot C_{in} \cdot C_{out} \tag{4.4}$$

Every convolution is followed by a batch normalization and a ReLU activation.
The other important structure used in this network is called a *downsampler block*

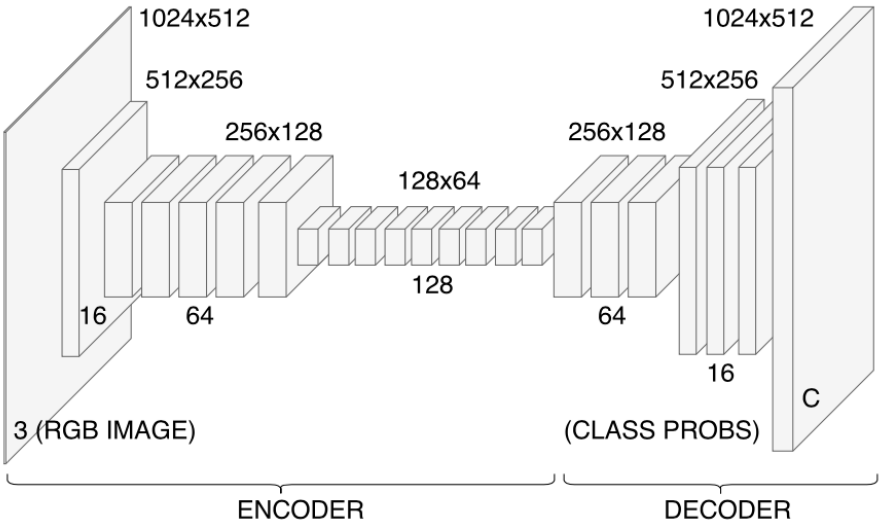
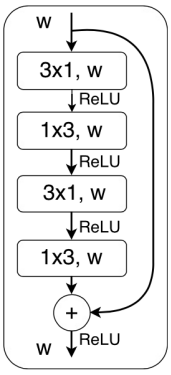
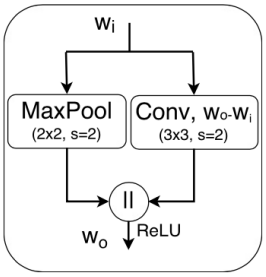


Figure 4.6: Encoder - decoder structure created by ConvNet

which is also explained in figure 4.7b. This type of downsampling was introduced by the ENet [14]. This block concatenates the feature maps of a 3x3 convolution and a max-pooling layer in parallel. It divides the spatial size by half what is applied in the whole network three times.



(a) Specific non-bottleneck structure



(b) Downsampler block with parallel pooling and convolutional layer

Figure 4.7: Types of blocks used in ConvNet

Table 4.7 shows the complete network with its layers and their output maps. This model uses an early downsampling of the spatial size of the feature maps to make efficient use of its layers. The depth of the feature map changes every time and just then when the spatial size gets changed. So this model just operates on three different sizes of the feature map.
The main part applies eight non-bottleneck blocks with different dilation ratios. The dilated convolutions in this network replace further downsampling of the feature map to gather more context. Through the bigger receptive field of its convolutions, this model learns global context without decreasing the spatial size.

Layer	Type	out-C	out-Res
1	Downsampler block	16	128x256
2	Downsampler block	64	64x128
3-7	5 x Non bottleneck 1D	64	64x128
8	Downsampler block	128	32x64
9	Non bottleneck 1D (dilated 2)	128	32x64
10	Non bottleneck 1D (dilated 4)	128	32x64
11	Non bottleneck 1D (dilated 8)	128	32x64
12	Non bottleneck 1D (dilated 16)	128	32x64
13	Non bottleneck 1D (dilated 2)	128	32x64
14	Non bottleneck 1D (dilated 4)	128	32x64
15	Non bottleneck 1D (dilated 8)	128	32x64
16	Non bottleneck 1D (dilated 16)	128	32x64
17	Deconvolution (upsampling)	64	64x128
18-19	2 x Non bottleneck 1D	64	64x128
20	Deconvolution (upsampling)	64	128x256
21-22	2 x Non bottleneck 1D	64	128x256
23	Deconvolution (upsampling)	19	256x512

Table 4.7: Layer disposal of ConvNet

Instead of the commonly used fast upsampling this network proposes a quite slower upsampling compared to the little number of layers used for this network. Furthermore, it does not use simple upsampling but the computationally more complex deconvolution with stride 2.

Results ConvNet

This part presents the results of training the convnet on the cityscapes dataset. As before this model is also trained on the cross-entropy loss and with the accuracy and mean IoU metric. The adam optimizer controls the learning process and the model is trained over 1000 epochs with a batch size of 2.

Metric	Result
Pixel accuracy	0.9151
Mean IoU	63.68
Runtime [s]	0.3388

Table 4.8: Segmentation results if ConvNet

Table 4.8 show an accuracy of 91.51% and an mean IoU of 63.68%. Here it happens

that the accuracy is higher than with the Fast-SCNN but the mean IoU is lower. This states an less consistent recognition of the different classes in this model. As well the runtime is higher than the other models. This is an effect of the computationally expensive dilated convolutions used in this model. Generally this model reaches also very good results for real-time models.

Figure 4.8 shows the training process of the convnet over the 1000 trained epochs. The figure shows on the left the accuracy and on the right the loss value.

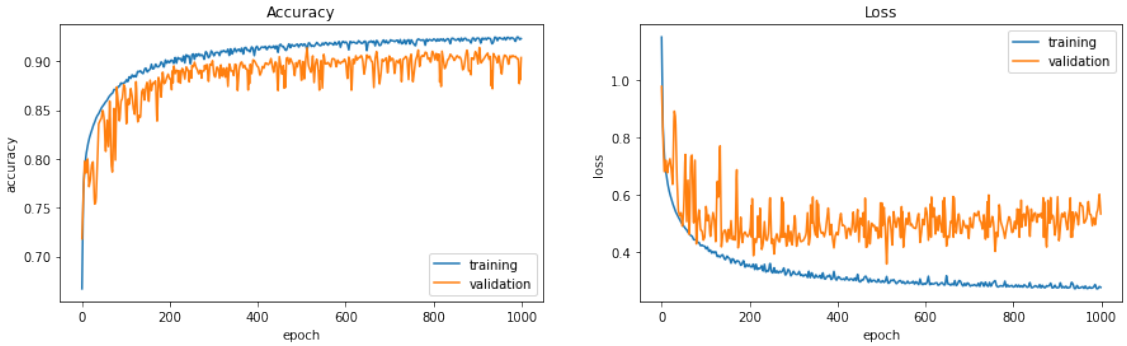


Figure 4.8: Accuracy and loss of training with ConvNet over 1000 epochs

Especially the loss function shows that the model tends to overfit on the training data. Where the loss on the training set rises continuously it starts to increase again after 300 epochs on the validation data. Moreover, it contains a high variance in its results because both curves are oscillating.
On the other hand the accuracy is raising during the complete training although the loss begins early to increase again. The final accuracy on both sets is very close and reaches nearly 92%.

4.2.4 Custom SCNN

This part presents a new model which surpasses the others in terms of accuracy and runtime. This customized model is mainly based on the Fast-SCNN presented in section 4.2.1. The Fast SCNN is extremely efficient and still achieves very good results. Due to its performance which is faster than real-time, there are applied little changes to increase performance at little costs of runtime.

The first change is made not due to the increase of performance or runtime but just to make the model better accessible. The original one uses for the Pyramid Pooling Module (PPM) a lambda layer to call an external resizing function inside a network. This function cannot be loaded, so the model cannot be either. It could be used by constructing the model and loading its weights but this change makes it possible to save and load the model on its own.
Therefore an upsample layer is used instead of the resize function in the original network. These to do the same by just resizing the different tensors to the final

output shape. The only difference applied is found in the pooling sizes. The original network uses a MaxPooling operation followed by a convolution to reduce the shape from 32x64 to spatial sizes of [1,2,3,4]. Every but the third can be restored by a pooling layer to the original size of 32x64. The feature map of 3x3 cannot be extended to the output map of 32x64 by any upsampling layer. That is why the custom version uses pooling steps to spatial sizes of [1,2,2,4] which can be easily extended to the original shape by an upsampling layer because these do not support decimal numbers. Moreover, this operation is more efficient than loading an external function.
Another performance improvement can be made in the upsampling path. Here the original network just uses an 8x8 upsample layer which does not contain any weights to upsample the image from the solution 128x256 up to 1024x2048. Here the new model uses two transposed convolutions to upsample the feature map step by step while applying weights to the upsampling. These help to sharpen the predictions and learn the boundary context. The depth of the feature map gets downsampled in three steps than just in one. These steps increase performance further at cost of more parameters which leads to longer runtime.
As well as the upsample part the downsampling part can be adjusted as well. Here the original model uses separable convolutions due to efficiency reasons. While normal convolutions have been proven useful in the first few layers these separable convolutions are changed by normal ones. These as well increase the runtime a little due to more parameters. The comparison of both models regarding their number of parameters and their runtime is shown in table 4.10.
The added convolutions bring a few more weights to train what makes this model more susceptible to overfitting. To avoid that there are implemented some dropout layers before the convolutional operations. These are especially important in the new down- and upsample branches.

The following table 4.9 presents the different improvement steps and their effects. First, every change gets evaluated separately and after that, the results of the combined changes are presented.
The default model reaches an accuracy of 0.9055 on test data and shows a balanced training, were training and test accuracy are very close. Here the model reaches his limits because accuracy does not increase any further by training over more epochs. The training of this model is tested for 1000 epochs because more training will not improve performance any further. However the model does not suffer from overfitting on this stage.
As experiment 1 shows, just adding dropout layers does not push the network performance. It is used a dropout of 0.3 before the convolutional blocks. The experiments show that the accuracy of test data does not increase compared to the default network. Nevertheless, it improves on training data. These experiments state the expectations that the model does not overfit on the training data but it cannot enhance the results. This technique will get more useful when applied together with other adjustments.

No.	Dropout	Downsample	Upsample	PPM	Train	Test
-	no	DSConv	Upsample	Lambda	.9102	.9055
1	yes	DSConv	Upsample	Lambda	.9242	.9047
2	no	Conv	Upsample	Upsample	.9276	.9061
3	no	DSConv	Conv	Lambda	.9369	.9100
4	no	DSConv	Upsample	Upsample	.9080	.9049
5	no	DSConv	Upsample	Upsample*	.9019	.9074
6	yes	DSConv	Conv	Lambda	.9254	.9042
7	no	DSConv	Conv	Upsample*	.9316	.9098
8	no	Conv	Conv	Upsample*	.9465	.9109
9	yes	Conv	Conv	Upsample*	.9260	.9043

Table 4.9: Improvements applied to create custom SCNN

The usage of normal convolutions instead of depthwise ones increases the performance a very little bit. Furthermore, the model does start to slightly overfit after around 800 epochs. Further training here will just increase performance on the training data.
On the other hand the influence of slower upsampling by transposed convolutions increases the results very much. In experiment 3 the accuracy on both sets rises significantly. This states that slower upsampling by convolutions is beneficial for this model. Here the model reaches the overfitting part earlier but as well the performance on test data does not stagnate too early.
As expected the experiments of using upsample layers instead of a resize function in the lambda layer do not change the results significantly. Two experiments no.4 and no.5 define different interpolation methods in the upsampling layer. The one marked with the * uses nearest neighbor upsampling and the other one a bilinear interpolation. The bilinear upsampling approximately reaches the same results as the default model which as well uses bilinear upsampling in the resize function. With the nearest neighbor method, these results are surpassed on the test data. This means the change of the pyramid pooling steps does not influence performance as strong as the interpolation method.

The next part describes the results of new models with the combination of the changes presented in table 4.9. The best model is the one with slower upsampling which still suffers from overfitting. To prevent this some dropout layers are applied before the convolutions in the upsampling part. Experiment 6 shows that dropout in the upsampling part harm performance on both data sets instead of slowing down convergence towards the training data and better generalization on test data.
Combining the upsampling by convolutions with the different PPM results stay equal to the ones with the default PPM. Here the upsampling layer with nearest-neighbor interpolation does not bring an improvement. At least it reaches less convergence

towards the training data.
By also adjusting the downsampling part by changing the depthwise convolutions to normal convolutions the performance of the model can be improved again. Here the accuracy of test data exceeds all the other models. As well on the training data the accuracy rises. Due to the new parameters, the model can learn more complex and can improve the performance of the dataset. Similarly, the overfitting increases which leads to a bigger difference between the results on training and test data.
To prevent this the usage of overfitting gets evaluated again. Every convolution to downsample the feature maps, in the beginning, follows a dropout layer with a dropout ratio of 0.3. Experiment 9 shows that the dropout again harms the results and cannot prevent overfitting.

The training progress is shown in figure 4.9. It shows the progress of the accuracy metric on the left as well as the loss on the right.

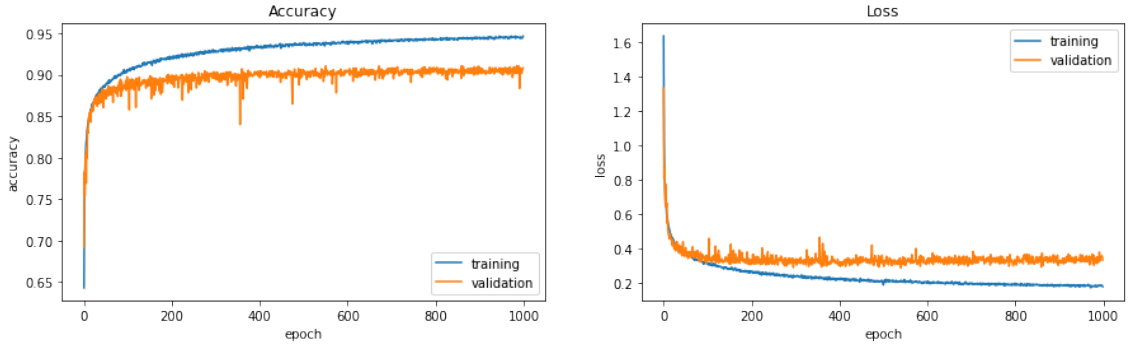


Figure 4.9: Accuracy and loss of training of SCNN over 1000 epochs

This model is clearly facing the overfitting problem. Where the accuracy on training data monotonously rises stagnates the accuracy on the validation data and just increases a very little bit. The loss looks similar and stagnates at 0.4 on the validation data.
Compared to the training progress of the original fastscnn shown in figure 4.2 this new network faces more problems of overfitting where the original network reaches its limits. Here the accuracy on the training data increases higher than on the validation set (like shown in table 4.9).

The next part evaluates the runtime performance of the original Fast SCNN and the two most influential improvements made to it. Fast SCNN * describes the model in table 4.9 referred to as experiment 3 and Fast SCNN ** describes the model in experiment 8.
Due to the changes applied above the runtime increases a little bit. The runtime is evaluated by averaging the prediction time of 100 random examples of the test set. The model runs on a Titan Xp with 2GPU.
The results show that for the Fast-SCNN the runtime increases proportionally to increasing parameters. The original one is the most efficient but also runs faster

	Parameters	Runtime	Accuracy	Mean IoU
Fast SCNN	1.817.162	0.2524	0.9100	66.42
Fast SCNN *	1.883.700	0.2744	0.9137	69.66
Fast SCNN **	1.920.772	0.2984	0.9109	68.76

Table 4.10: Runtime evaluation of Fast SCNN and its improved version

than real-time. BY just give up a little runtime an improvement of nearly 5% of mean IoU can be made. This resumes that the Fast-SCNN* from experiment 3 in table 4.9 is the best model of these three.

As mentioned before the most effective metric to evaluate an model for the segmentation problem is the IoU or the mean IoU. The next part evaluates the difference in IoU of both models which is presented in figure 4.10. It shows the IoU for every class for the Fast-SCNN 4.2.1 in blue and the SCNN 4.2.4 in orange.

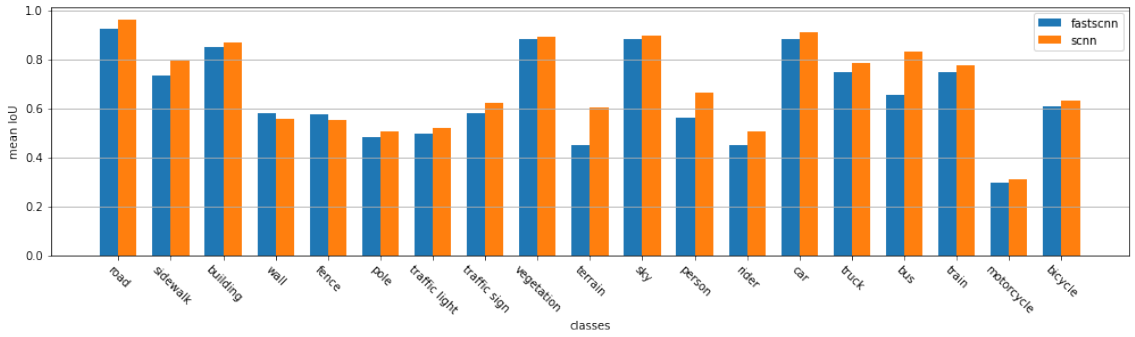


Figure 4.10: Comparison of SCNN and original Fast-SCNN regarding IoU

Like indicated in the evaluation of the SCNN the performance of both models is very close. This also gets proven for the IoU by figure 4.10.
In general the distribution shows a little difference between classes that are detected well and others that are less often classified correctly. The good classes are the road, the sidewalk, buildings, vegetation, sky, and cars. Classes that are hard to classify are for example poles, traffic lights riders, and motorcycles. Noticeable, that these classes are all very small objects and do not appear in big pixel groups. The other objects like the sky, the road, buildings, or vegetation are contradictory and appear mostly in bigger pixel groups. This leads to the conclusion that bigger objects and classes are most likely easier to detect. This gets proven by the point that the boundaries of objects are the most difficult part to classify correctly. Smaller objects naturally contain more pixels of area boundaries.

The Fast-SCNN surpasses the SCNN on two classes (wall and fence) where the SCNN performs better on the other classes. The biggest differences can be detected for the classes terrain, persons, and riders. This does not show any trend towards

smaller classes or smaller objects which might be detected worse with worse models. The classes with the biggest difference in IoU seem to be arbitrary and can be justified with the variable training behavior of neural networks. The class with the best results are for both networks the road and cars. Lowest IoU accuracy is detected for motorcycles.

4.3 Evaluate Training methods

This section describes the different experiments made in the training phase. Experiments are made regarding the number of training classes, data augmentation techniques, pretraining, and image resolutions. All the effects are shown on the customized Fast-SCNN from section 4.2.4.

4.3.1 Compare with training on more specific classes

This part compares the results trained on different segmentation classes. The original dataset specifies these images much more specific than it is useful to train on because little classes have fewer appearances. Because of this, it is better to group them. Moreover to mix different data sets their classes must match.

Figure 4.11 shows the training progress of the SCNN trained on the cityscapes set with 34 classes. The figure shows on the left the progress in accuracy and on the right the change of the loss value. For this example, there is no data augmentation applied.

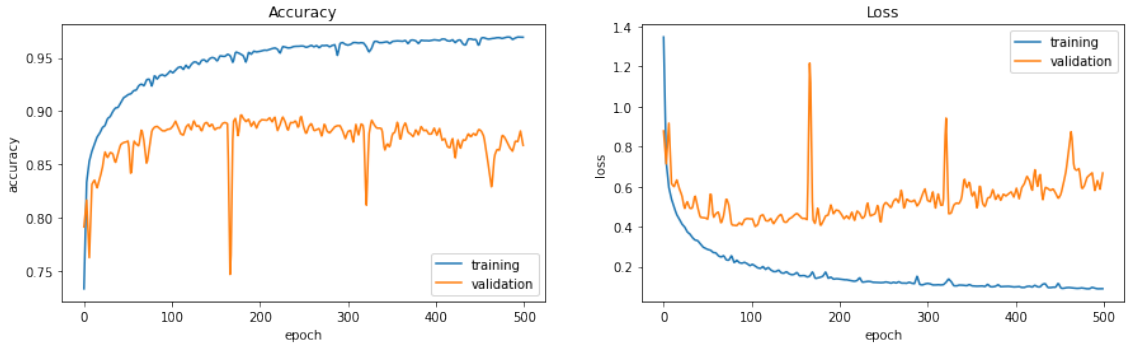


Figure 4.11: Accuracy and loss of training of SCNN with 34 classes over 500 epochs

It shows that it tackles the problem with overfitting much stronger than the models trained over the 20 classes. The loss begins to increase again just after 200 epochs. The accuracy as well starts to drop over time. It reaches its maximum after 200 epochs. As well, the training progress is not that smooth and contains some outliers where the performance drops a lot for just one epoch. The performance on the training set is very good and reaches 96% of accuracy. This indicates that the model is sure capable of learning to classify the scenes in more classes but it suffers under

big overfitting towards the training data.

Table 4.11 compares the accuracy and mean IoU of the model trained on all 34 and on the recommended 20 classes. The election of these classes was introduced in chapter 4.1.1.

	34 classes (original)	20 classes (recommended)
Pixel accuracy	0.8983	0.9137
Mean IoU	41.91	69.66

Table 4.11: Evaluation of different number of segmentation classes

It shows that the accuracy reached with all 34 classes is very high and close to the one trained on just 20 classes. It nearly reaches 90% accuracy. The mean IoU shows the downside of training on that many classes. The IoU just reaches 41.91% To be able to compare them better the average over the 20 recommended classes is taken for this value. Here the results indicate that there are lots of classes which are just very barely classified correctly. These are mainly the small classes. This leads to the high accuracy but lower IoU.

Figure 4.12 evaluates the IoU on the 20 recommended classes. It shows the consequences of training on more detailed classes and the effects of usefully grouping them. The performance with all 34 classes is represented by the blue bars and the orange bars represent the results obtained with the model trained just on 20 classes.

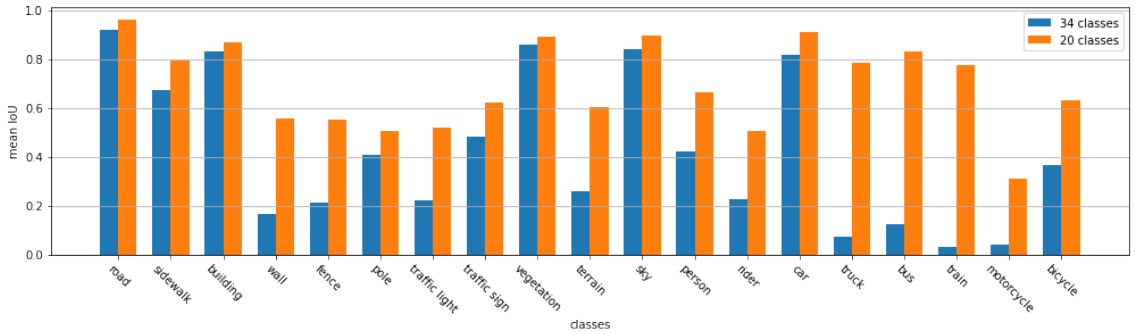


Figure 4.12: Comparison of mean IoU of training on 19 or 34 classes

The figure shows that the model trained on all classes compared to the one trained on 20 classes results in very uneven distribution. Some classes are classified very well but others are classified very badly as well. This difference is also there for the 20 classes model but with 34 classes it is far bigger. Wherewith 20 classes every class except motorcycles are classified with at least 45% with 34 output classes there are 8 out of 20 classes which are detected with equal or less than 20% of IoU. These mainly are classes that do not appear very often like trucks, buses, trains, or

motorcycles. As well there are the main classes like the road, sidewalks buildings, vegetation, sky, or cars that are classified nearly as good as with the other model trained on just 20 classes.
This classification pattern underlines the results shown in table 4.12 that the accuracy is close but there is a bigger difference in IoU between the models. Due to the classes with low IoUs the mean IoU is also low.

The following part evaluated the three class groups separated with the model trained on 34. It compares the IoU to analyze the sense of purpose of grouping these recommended classes together. The IoU over all classes, as well as the IoU over the 15 classes which are ignored by the grouping step, is shown in table 4.12.

Classes	all (34)	recommended (20)	ignored (15)
Mean IoU	32.30	41.91	20.11

Table 4.12: Evaluation of mean IoU on different classes

Table 4.12 shows that on the 20 recommended classes the iou is with 41.91% far higher than on all classes where it only reaches 32.30%. The classes which are ignored in the training just reach an average of 20% IoU. This can be misleading because also the ignored classes contain labels that are very well classified. These are for example the ego vehicle or the rectification border. These are the outer pixels with which every image is padded. All input images contain a 3 pixel frame around the real image.
The IoU for all 34 classes is presented in figure 4.13. The red marked bars are the labels that should be ignored during training and the blue bars are the classes to normally train on.

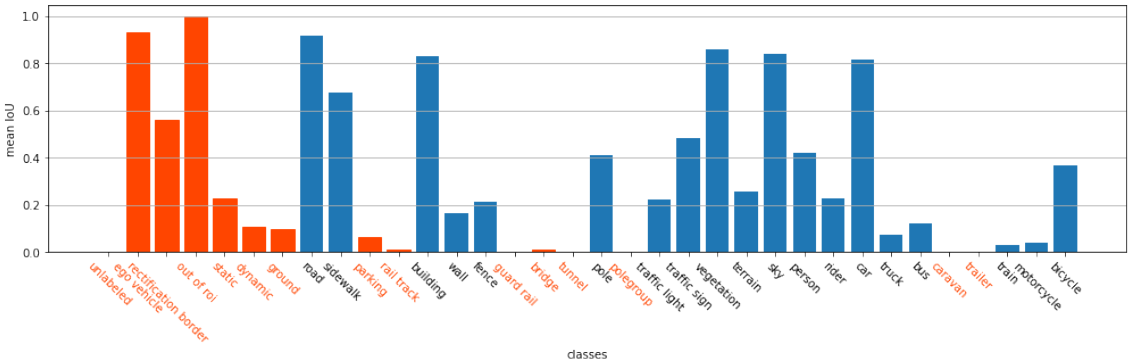


Figure 4.13: Evaluation of IoU on all 34 classes

Noticeable, that besides the three big ignored classes (ego vehicle, rectification border, out of ROI) which are detected very well, the other ignored classes return very bad results. These are indeed classes that firstly do not appear very often in the dataset and second do not have any specific characteristics for autonomous driving.

For example, analyzing the vehicles there are cars which is the main group, trucks which are bigger and slower, buses which are important because these might are allowed to drive on specific lanes, trains which ride on rails and the two-wheelers. On the other hand caravans and trailers do not have any specific characteristics why these need to be classified especially.

4.3.2 Compare data augmentation techniques

The experiments in this part focus on various data augmentation techniques. These are used to increment the variety of training data and make the model generalize better. For the segmentation task, there cannot be used any augmentation technique to improve results. Some techniques work better where others are completely unsuitable.
The data augmentation techniques are just applied to the training data and the validation data is not touched during this process.

The most common and basic one is randomly flipping the image. For the neural network, the image changes completely and creates a new training example. For this task of urban street scenes, vertical flips do not make sense so just horizontal flips are applied. It is not helpful to learn images upside down because e.g. cars, trees, or buildings are mainly located on the bottom half of the image and look completely different upside down. Furthermore, they do not appear in the test set upside down so the vertical flipping is not applied. On the other hand, horizontal flips are a very useful technique because they do not change the scene and the images do not get more dissimilar to the test set. If the image is flipped the corresponding label needs to be flipped too.

A very useful technique for segmentation tasks is every sort of noise applied to the original image. Changing randomly selected pixels is a good way to avoid overfitting although it is not that effective as other techniques like flipping. Because the labels are feature maps these techniques must not be applied to the labels. Changing values of the labels leads to inconsistent training because the same object now gets assigned, different classes.
Most natural types of noise are brightness changes and channel shifts. Brightness changes are defined by an interval and for every image, in the training batch, there is randomly selected one different value to change the brightness of the data. A brightness value of 1 means the original image. While augmentation there can be used values smaller than 1 as well as values bigger than 1. This makes the image look brighter or darker.
As well as brightness changes, channel shifts are used to randomly change the pixel values of the input images. In this case there randomly are added values inside the specified interval to every image channel. These values which get added can be positive as well as negative and are the same for every pixel and every channel.
Pixelchanges like Gaussian noise, blur, contrast changes, saturation changes are not

supported by Keras. Also in the state-of-the-art networks, nothing different than the two mentioned are used. For this reason, this work focuses on these two pixel-wise augmentation methods.

A very helpful technique is zooming into or out of the image to create new samples. Zooming into the image means cropping any random part and resizing the cropped image to the original size. This crop keeps the spatial dimension to not change the proportions of the displayed objects.
An important detail by zooming in is to specify the desired interpolation methods. The interpolation method for the image can be chosen freely, which one fits the best for the model (bilinear, cubic, gaussian, etc.) For the label, on the other hand, it necessarily has to be the nearest neighbor interpolation because every value in the label feature map describes a specific class. Bad interpolated classes in the label result in unclear contour in the final predictions.
Zooming in makes every object seem to be bigger but keeps the scene basically as it is. Otherwise zooming out displays the image smaller and pads the undefined parts with new values. The interpolation methods to define the new interpolated values due to zooming stay the same as if zooming in. For the new generated pixels outside the original image it needs to be chosen one filling method out of *nearest*, *constant* or *reflect*.
Nearest pads every outside value with the last known image value. Because every image in the dataset is surrounded by three pixels of 0 the outer pixels theoretically are be filled with zeros. Due to shifting or zooming these outer pixels can be removed. Because of this case, the images can be padded by any value existing in the image data. This case appears very often by applying aggressive data augmentation. This method is very useful for little translations.
By choosing the constant mode the padding value can be chosen and will be constant over every padding. This option is specially designed for images taken over monochromatic backgrounds but is less useful for natural images.
The last option is to reflect the last values of the image. This method is useful for continuous or natural backgrounds containing trees, mountains, houses, etc.

Shifting is another affine transformation that can be used to create new image data out of the original ones. Shifts can be applied horizontally or vertically. For this work, there are used both because they do not change the scene. As well as zooming out, shifting creates new values outside of the image. These values are filled by the same method as for zooming.
The combination of cropping and zooming gives more variety to the generation of training data instead of just centered zooming. Due to the shifting, it can be zoomed to any corner too.

Table 4.13 presents the experiments made to evaluate the different augmentation methods to reach the best results. Applied are the just described methods with various parameters. This part does not include the combination of data augmentation

methods and evaluates each one separately. Due to the high training costs, these methods are evaluated for 100 epochs to identify tendencies and set up the right combination later on.

Data Augmentation					Accuracy	
shift	bright	zoom	channel	flip	train	test
-	-	-	-	-	.9509	.9001
.2*	-	-	-	-	.9357	.8840
.2**	-	-	-	-	.9171	.8886
-	[.5,2.]	-	-	-	.9357	.8892
-	[.7,1.5]	-	-	-	.9386	.8904
-	-	0.5*	-	-	.9246	.8805
-	-	0.5**	-	-	.9121	.8868
-	-	[1.,1.5]**	-	-	.9300	.8804
-	-	-	20.	-	.9499	.8965
-	-	-	-	x	.9413	.9010

Table 4.13: Effect of different data augmentation techniques applied separately

The results without data augmentation seem to be better than the rest. Just with flipped images better accuracy is reached. But the difference between the known and unknown data is quite big. Here the model reaches its limits and cannot improve further.
Figure 4.14 shows the training progress of the presented data augmentation techniques. Without any augmentation, the accuracy on training data is strictly monotonously increasing and reaches very fast an accuracy of 0.95 while on the test data the accuracy just slightly passes 0.9. Anyway, the accuracy is still increasing on both sets which means that the model still not has reached its limits.

The results of training with flipped images exceed these results and reach the best results. Due to the increased training data and variety due to the augmentation the accuracy on the training data also does not rise that high like without. This leaves room for further improvements by training over more epochs.
The training progress with flipped images looks nearly the same as without augmentation but reaches over the whole training better results. An important difference is that the results on the training data do not increase that fast and converges slower. As consequence, the training can be kept up longer which pushes the overall results.

By applying channel shifts the training reaches good results in accuracy compared to the other ones. On the other hand are the results on both sets of data worse than without data augmentation. This indicates that it harms performance on the test data while not slowing down convergence towards the training data overfitting. Still, it proves to be a valid method to increase variety in the data and increase results when it is combined with other methods.

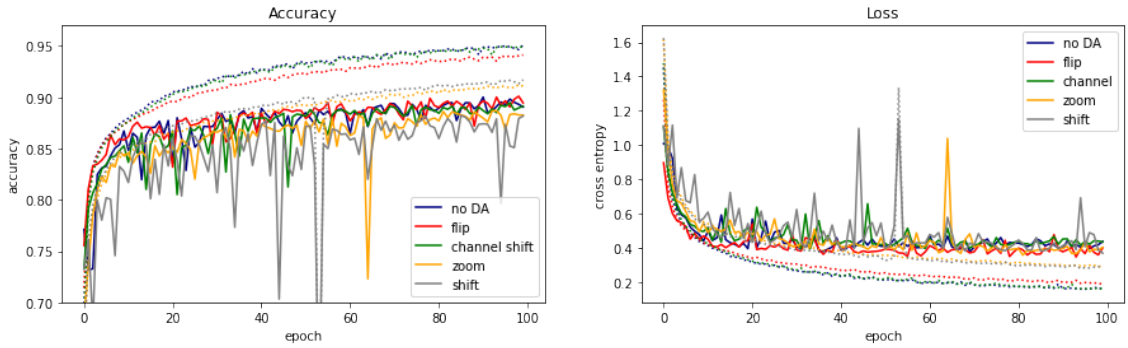


Figure 4.14: Comparison of training of different data augmentation techniques

The training progress shows that it does not drastically slow down learning but accuracy on the test data increases stronger than with the before presented methods. This enables effective training over more epochs. Apart from that the results on test data are less consistent and every once in a while it produces outliers with bad prediction results.
Applying brightness changes as well does not prevent overfitting on large scale but it postpones strong convergence towards training data and does not harm performance on test data very much. This indicates it will be useful for long training but it needs to be combined with other techniques. On its own as well as channel shifts their effects are rather small.

To prevent or postpone overfitting shifting the original image in the x and y direction is a very useful method. The accuracy on both data sets does not rise that fast and strong as without. Also, the performance on the test set increases slower. These experiments here use a shifting of up to 20% in both directions with either padding of reflecting pixels or extending the nearest one. Especially with the reflecting pattern the accuracy on both data sets is very close which indicates good training without any tendency to overfit. On the downside, this method slows down training very much.
The big effect of applying shifts is shown by its training curves which are still very close to each other and continuously increasing. The model is not even close to reaching its limits. On the other hand, shifting increases the inconsistency over the training and produces outlier results.

Zooming is a more aggressive data augmentation method too. The results show less convergence towards the training data which indicates that the training volume can be increased further. The same applies to the results on the test data. Here the results are worse than without augmentation after the same number of training epochs. The table shows the comparison between randomly zooming out and in or just zooming in. The results are better by randomly applying in/out zooming because the accuracy on test data is higher and lower on training data. This lets room for further training and counteracts overfitting.

As well it is shown that the reflecting pattern is way more useful than the nearest respectively the constant mode. The nearest neighbor padding results in a constant zero padding around the images by zooming out. By zooming in there is not used no padding. These experiments state the expectations that reflecting the outer pixels proves useful for natural images like urban street scenes.
The training curves look the same as with shifted images. As well it does not overfit on the training data and it increases continuously but it also produces outlier results. But for the performance, these are not important and can be accepted.

Combined these methods influence one another and boost their performances. The effect of avoiding and postponing overfitting is shown by the number of epochs trained to reach the limits of the network.

		Data Augmentation					Accuracy	
no.	epochs	shift	bright	zoom	channel	flip	train	test
0	200	-	-	-	-	-	.9548	.9012
1	300	-	[.5,2.]	-	20	x	.9555	.9102
2	500	.1*	[.7,1.5]	.3*	10	x	.9486	.9041
3	500	.1**	[.7,1.5]	.3**	10	x	.9437	.9077
4	1000	.2**	[.7,1.5]	.3**	10	x	.9492	.9081
5	1000	.2**	[.5,2.]	.5**	20	x	.9426	.9137
6	1000	.2*	[.5,2.]	.5*	20	x	.9449	.9087

Table 4.14: Effect of joined data augmentation techniques applied

Flipping will be applied in every step because it is the only technique which also separately improved results. As well it can be naturally combined with every other technique without harming the effect of another one. In the first step just pixel scaling data augmentation techniques are applied together. This boosts the accuracy by nearly 1%. There are applied very aggressive values for brightness changes and channel shifts. The results show that the accuracy on the training data rises slightly but a lot on the test data. Here it can be concluded that these transformations does not prevent effectively overfitting on the training data because the accuracy is still far higher than on the test data. But it converges slower so the model can be trained over 300 epochs and which improves the performance on the test set.
For experiment 2 there are less aggressive pixel scaling techniques but also shifting and zooming applied. It uses the nearest neighbor padding (*) for pixels outside the image. Here the accuracy does not raise that fast and it can be trained over more epochs. Although the accuracy of the training set is very high, the accuracy of the test data improves just a little compared to the results without data augmentation. By changing the padding mode to the reflect model (**) the performance on the test data increases and decreases on the training data. This confirms more generalization and also slows down convergence towards the training data. These results do not reach the ones from experiment 1 but it leaves room for longer training.

By increasing the maximum shifting factor and training for longer, the results again improve a little. Especially the performance on the training data increases and the model reaches stronger overfitting.
Increasing the data augmentations by stronger shifting and zooming this can be limited. Here the model does not reach that strong overfitting towards the training data like in the experiment before but also improves the performance on the test set. This little change leads to the best results with combined data augmentation techniques.
Compared to them the results with the nearest neighbor padding drop slightly. On the training data, it is still quite high but on the test data, it lacks performance. This underlines the assumption that for realistic images the reflecting pattern works best and outperforms the nearest or the constant mode.

To conclude the effect of data augmentation techniques, for this task is has proven to be very effective. The main reason is to postpone overfitting on the training data and enable more training. This effect can especially be shown in table 4.14. Here the training size is five times larger due to adding different techniques.
Furthermore this section describes the effect of each technique. Methods like shifting or zooming slow down the convergence towards training data very much. The slower training can particularly be observed in the results in table 4.13. Here the results seem worse than the others because at this training step it does not reach results as good as without. With bad parameters, the overall effects are very small and can lead to local maxima. But used in the correct way it is useful to improve the overall performance. Especially the padding pattern has a big influence on the training. This effect gets greater the larger the training size is what gets clear in table 4.14.
On the other hand the pixel changing techniques not just slow down the training a little bit but pushes the results on the test set quite well. This is indicated by the results in table 4.13 and proven by the results in table 4.14.

4.3.3 Evaluate influence of pretraining

Another method applied to increase performance is to pretrain the models with another dataset. Pretraining can have a good influence on the performance of neural networks and help to find global maxima. For pretraining, the model is trained on the mapillary vistas dataset 4.1.2. To present the effects of pretraining the customized SCNN 4.2.4 is used. It first shows the performance of the model on the pretraining dataset and afterward the effects of using this as pretraining.

Pretraining on mapillary vistas dataset

First, the model is trained on the mapillary dataset. This one got far more labels than the cityscapes set so here the labels got converted like shown in section 4.1.2 in table 4.2. This is done to match the classes and got the same labels on the same objects.

The balance of both data sets is compared in the following part to identify differences. Figure 4.15 shows the percentage number of pixels in both data sets. The blue bars are the number of the pixels in the mapillary set and the orange bars the ones in the cityscapes set.

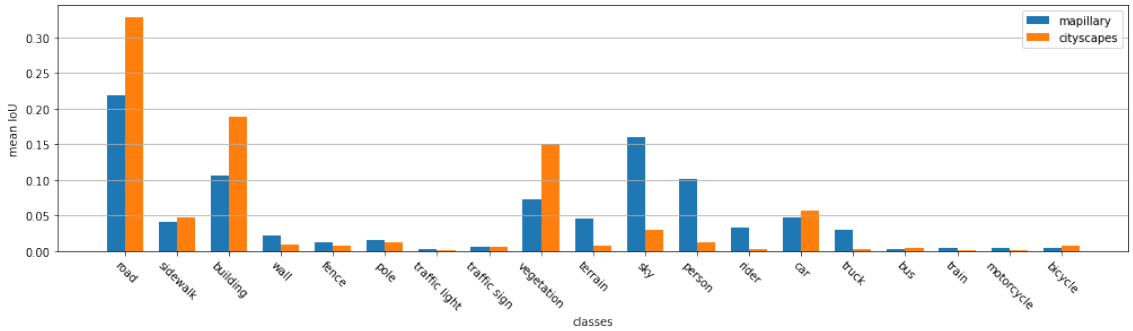


Figure 4.15: Pixel occurrence of classes in the cityscapes compared to mapillary vistas dataset

First the figure shows a similar distribution for all classes on the two data sets. It also gets clear that the camera angle how the images are taken is steeper for the cityscapes set. This leads to more pixels belonging to the road and consequently for the mapillary set to the sky. As well the cityscapes data is limited to cities where the mapillary set includes scenes outside of cities. This got shown by fewer pixels of buildings and vegetation. As well the number of pixels of the class persons is significantly higher in the cityscapes dataset. This might help to especially pretrain on this class. The other classes are very even.

Another important point is that the dataset is 6 times bigger than the cityscapes set so the training lasts 6.7 times as long. This makes it difficult to train it over a long period as 100 epochs run for over 12 days straight. Because of this, the model has just trained over 100 epochs which are shown in figure 4.16. The left graphic shows the progress in accuracy and the right one the progress of the loss function value. This model as well is trained on cross-entropy and uses the adam optimizer to adapt weights.
The training progress shows that the accuracy of training and validation data are very close. The one on the validation data is slightly higher but very close. On both sets, the accuracy is monotonously rising what indicates that it is training very well without overfitting towards the training data at this point. The training progress up to this point indicates that the limits of the model are still not reached and it will be useful to continue training this model.
As well as the accuracy the loss values indicate the same. On the validation set, it is slightly lower and both are dropping continuously. After 100 epochs the loss function nearly reaches 0.3.
In general the training process is very constant and is oscillating just a very little bit with very little amplitude. During the first 100 epochs, there are no outliers in

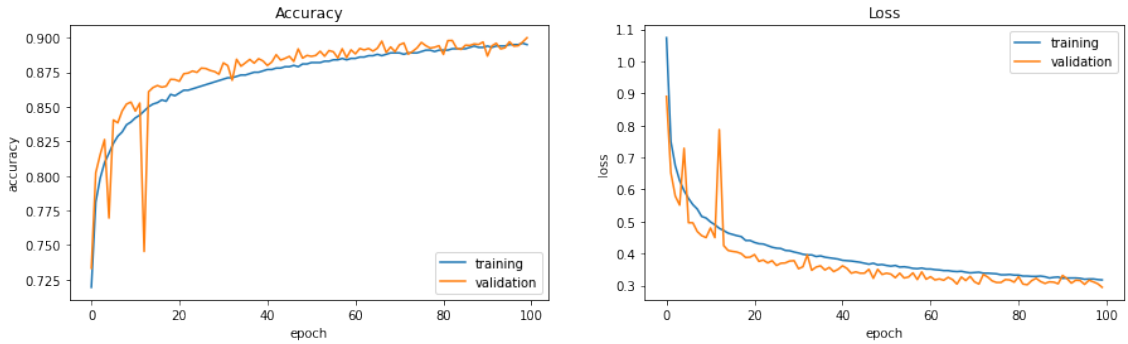


Figure 4.16: Accuracy and loss of training on mapillary data with SCNN over 100 epochs

the performance which means that the network adapts the weight very well.

The detailed performance of this trained model is shown in table 4.15. It presents the results of the loss value, accuracy, and mean IoU (training and validation set) on the mapillary dataset and compares them with the performance of this model on the cityscapes dataset. Important to note, that this model is just trained on the mapillary data and has still not touched the cityscapes set. Due to that the training set of cityscapes data serves as more validation data and it's a good way to get a bigger evaluation.

	Mapillary		Cityscapes	
Metric	Training	Validation	Training	Validation
Loss	0.3189	0.2950	0.4863	0.5045
Accuracy	0.8960	0.9000	0.8438	0.8346
Mean IoU	40.90	39.69	33.09	32.78

Table 4.15: Evaluate SCNN trained on mapillary on cityscapes data

The results of this table validate the good training on the mapillary training set shown in figure 4.16 with an accuracy of 90%. As well it shows that the performance for loss and accuracy on the validation set is still better than on the training data. In general the accuracy is very high which indicates very good results just after 100 epochs.
The mean IoU is very low compared to the good performance in accuracy and loss. This indicates that there is a big difference in IoU between the different classes. By detecting the classes which occur often in the dataset very well and just hardly detecting smaller classes the results can indeed return a very high accuracy although not all classes are detected well. This shows the importance of the Intersection over Union metric.
On the cityscapes data, the performance drops a little. The accuracy just reaches 84% and the loss 0.49. The mean IoU also drops a little and just reaches 33%.

Because these data belong to a completely different dataset these results still can be ranked as very good results. These results also must not be compared to the results presented in section 4.2 because this is pretraining on a different dataset.

Table 4.15 compares the mean IoU of the predictions on mapillary and the cityscapes date. To get a better understanding of why the accuracy is very high and the mean IoU is not, figure 4.17 shows the IoU on the mapillary and cityscapes validation sets for every class separately.

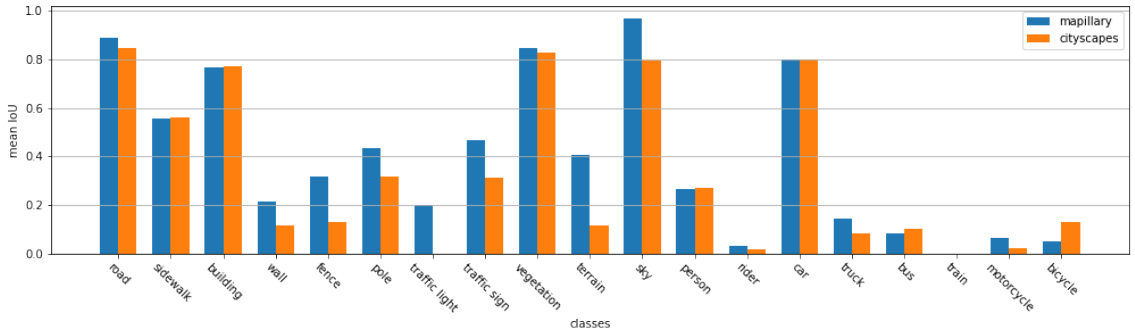


Figure 4.17: Compare IoU on mapillary and cityscapes data

The graphic shows that the model is indeed very good on the bigger classes like the road, buildings, vegetation, sky, and cars. The dataset contains lots of pixels that belong to these classes which pushes the accuracy drastically. Here the IoU is higher than 80% which is a very good value for image segmentation tasks.
On the other hand some classes are very difficult to detect. Examples for these classes are riders, trucks, buses, trains, motor- and bicycles. These are objects which do not occur often in the dataset and due to that, it is much harder to classify them correctly.
Comparing the results of the two data sets the performance on each class is very close to the other dataset. Mostly the results on the mapillary data are better. Exceptions to this are the classes building, persons, buses, and bicycles. The classes which the model detects better in the mapillary data are walls fences, poles, traffic signs and lights, and the sky.

Using pretraining on mapillary vistas dataset

The following training on the cityscapes dataset is performed over 500 epochs. Table 4.16 presents the results obtained without and with pretraining the SCNN on the mapillary set. It compares the performance without pretraining with the performance directly after pretraining and after pretraining and performing the main training. As evaluation metrics, the accuracy and mean IoU are used. The results without pretraining are obtained in the experiments presented in table **??** and the results after pretraining are shown in table 4.15 in this part. The results after applying the main training show that the results indeed improve a very little. The

Training method	Accuracy	Mean IoU
Without pretraining	0.9137	69.66
After pretraining	0.8346	32.78
Pre + real training	0.9145	69.67

Table 4.16: Improvements with pretraining on mapillary data

accuracy increases by 0.08 and the mean IoU by 0.01. This are very little improvements but it shows that pretraining on the mapillary data has a good effect on training.

The next part with figure 4.18 presents the training process after pretraining the model on the mapillary data. It shows the training over 500 epochs and the accuracy is shown on the left and the loss on the right.

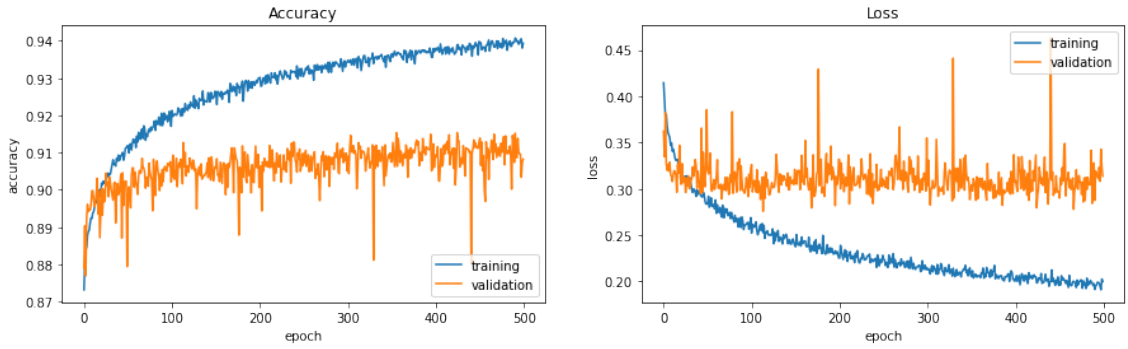


Figure 4.18: Accuracy and loss of training on cityscapes data with SCNN over 500 epochs after pretraining

First of all the values shows a big scattering while training. The values for accuracy as well as for the loss jump a lot between the following epochs. As well there are a few outliers down. This is typical behavior for models first trained on a very big, different dataset.
In general the training progress looks well and does not indicate critical overfitting towards the training data. The accuracy on the training data rises much faster and reaches significantly better results but as well on the validation data the accuracy is rising and performing very well.

4.3.4 Evaluate influence of image resolution

This part demonstrates the influence of the image resolution on the predictions. Lower resolutions can speed up the network but it needs to be evaluated how the model performance change.
This section compares three models trained on different resolution sizes. As lower resolution sizes a half (512x1024) and a quarter (256x512) of the original size

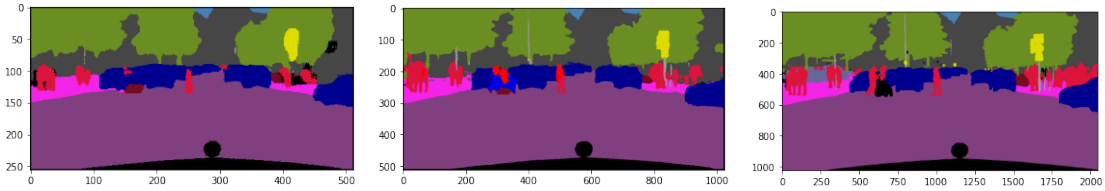
(1024x2048) are used for evaluation. The following table presents the results of accuracy, mean IoU and runtime of the three resolutions compared with each other.

	256x512	512x1024	1024x2048
Accuracy	0.8610	0.8911	0.9137
Mean IoU	39.16	46.45	69.66
Runtime	0.1134	0.1546	0.2744

Table 4.17: Compare results with smaller image resolution

As expected the results on the original resolution are the best in accuracy and mean IoU. Where the accuracy of the middle resolution is still good, with the small images it clearly gets worse.
This gets more obvious by comparing the mean IoU. Here the the performance drops drastically for the small images. These just reach 39.16%. Here the accuracy results can be pretty misleading because 86.1% does not look bad. The improvement of the half images also is small and insufficient because state of the art performance reaches over 60%. This is just reached by the original image size where also the accuracy reaches best results.
The runtime results show very big differences. By dividing the image resolution to the half the runtime as well is nearly just half as long. This is a very big change and way faster. The gap to the small resolution is not that big but it again drops significantly to almost a third of the runtime with the original images.

Figure 4.19 shows the predictions for one example image from the test set (never seen during training respectively validation). It compares the same prediction with the three models trained on every image resolution.



(a) 256x512 (b) 512x1024 (c) 1024x2048

Figure 4.19: Visual comparison of segmentation results on different image resolutions

By having a look on the visual results the predictions on the small images in figure 4.19a look very blurred compared to the others. It is not able to detect edges precisely (e.g. pedestrians). With the half of the original size this gets better but smaller objects still cannot be detected well. Small objects are just detected well on the predictions of the original images.
Another important critique is that objects of rare classes cannot be detected well.

For example with the model for small images poles are just not detected at all. These are too thin to make an impact on this resolution size. The pole here gets just classified as vegetation due to its background.
An example for classes that do not appear often is the wheelchair pushed by one person on position 500,650. Wheelchairs belong to the category of dynamic objects (defined in table 4.2) and should be marked with class 19 (black). These are hardly ever detected by the models with the smaller image sizes. Here it detects falsely motorcycles or again just background classes. The accuracy with small images is still quite high because the big classes (road, buildings, vegetation, sky, etc.) which occur much more often in the dataset are detected very well.
The model with half images predicts better contours of the recognized objects and the objects do not just look like stains of different colors. Here the scene can be imagined and as well small classes like the before mentioned poles are suggested. This shows a bigger improvement than the small raise of accuracy indicates.
The predictions on the original images show very clear contours and there are all classes detected very well. Even groups of persons are detected well and look very realistic. Furthermore, it is the only model where the wheelchair gets classified correctly.

This example of figure 4.19 proves why the accuracy is not an appropriate metric for image segmentation tasks and the Intersection over Union (IoU) is by far more representative which returns important details on the classification capacity of the different classes. The diagram shows the IoU on all 20 classes for the three resolution sizes. The blue bars represent the small resolution, the orange the half images, and the green ones the performance on the original images.

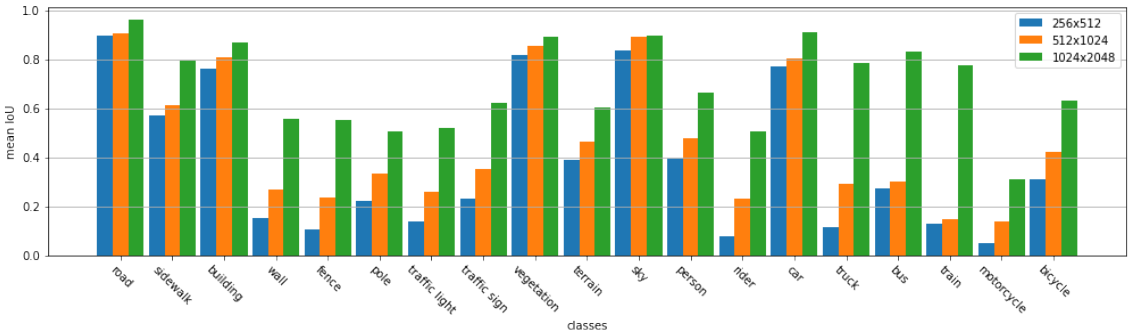


Figure 4.20: Comparison of IoU on every class

This graphic shows that the models with higher resolution get better for every class. The higher resolution got the best performance for every class and the quarter resolution is always the worst. Interesting is that on good detected classes, the difference of IoU performance gets smaller and the results are closer. As examples for that the classes road, building, vegetation, and sky are to mention. On these, the IoU is higher than 80% and for the small models, its results are not more than 5% worse. Especially the sky class returns nearly the same results on all three models. The

worst class for every model is motorcycles where the IoU is low with every resolution. Especially with the original resolution, the results on this class are worse than the other classes. The other classes which get detected badly on the lower resolutions, like a truck, bus, and train are better detected on the original resolution.
On the other hand on classes which are in general hardly detected the differences get bigger. The biggest differences are for the classes truck, bus, and train. The class train got a difference of over 60% IoU between the original to the small and middle resolution.
The difference between the different classes is less for the model with original image sizes than for the one with half or quarter image sizes. With the bigger images, the results on every class are much closer and nearly reach 60% on every class except motorcycles. With the smallest size, the difference in IoU of the best and the worst class is over 80%. At this resolution, it happens often that a few classes are very bad detected where others are detected well. This trend continues on the half-sized images.

4.4 Evaluation and Results

In this section, an overall evaluation of the described models and their results is presented. First, the SCNN presented in section 4.2.4 is evaluated in detail. Then all models are compared with each other and the advantages and disadvantages get identified.

4.4.1 Evaluate SCNN

As shown in section 4.2.4 the SCNN reaches good results but it still has the problem to detect a few classes. The following part analyzes the confusion matrix of its predictions. As introduced in chapter Theoretical Background, a confusion matrix compares the predicted classes of a model with its true classes. This relation is shown in figure 4.21. The original classes are presented along the vertical axis and the prediction along the horizontal axis. Every element indicates the number of appearances it is classified in a certain class. The values are normalized to every row so the predicted classes are shown in the relation to their true classes. The color bar on the right indicates the percentage values of all elements.
First of all the confusion matrix states the results shown in figure 4.10. The good classes are also here marked with high values. The confusion matrix is based on the accuracy metric and not on the mean IoU, but the results are still similar.
The most interesting part to observe here is which other classes of objects tend to be falsely classified. In general, the class is well classified if the diagonal element has a high value and the corresponding row and column have very low values. These areas already identify the classes cars, vegetation, and sky.
For example the diagonal element of the class building has a very high value but its column contains many elements bigger than 0. This means that lots of other classes are falsely classified as buildings. These are lots of classes and cannot be grouped

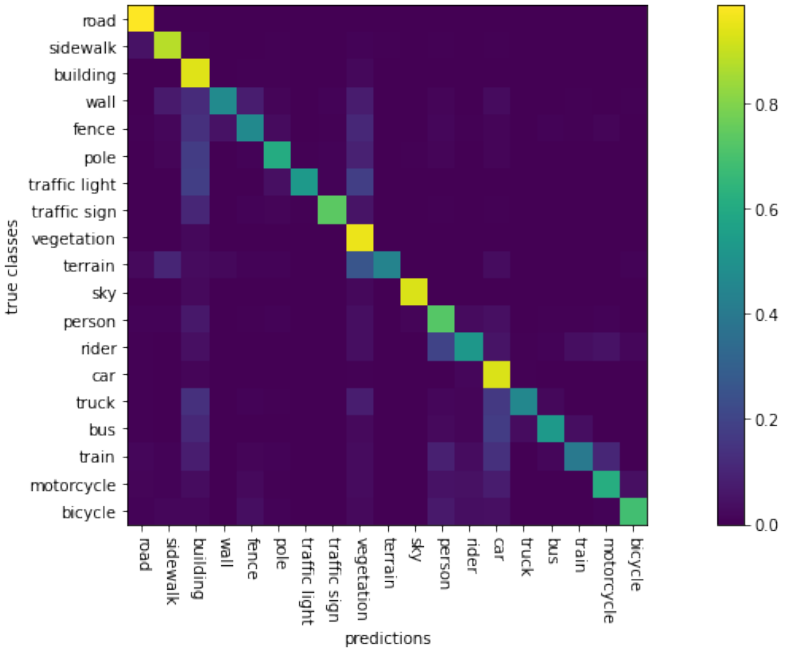


Figure 4.21: Confusion matrix of predictions of SCNN

into any general category. The problem here is that buildings are a background class where foreground objects sometimes cannot contrast.
The same happens to the class vegetation. This also contains some objects like poles or traffic lights which are wrongly classified because they might not contrast. It also shows that the terrain class is often classified as vegetation. Indeed these two classes have similarities because the terrain describes flat vegetation like grass.
The classes persons and riders show the same problem. Lots of riders are classified as persons and others as motorcycles. This makes sense because riders and persons are the same categorical group and riders just appear on two-wheelers.
Another systematic error occurs by classifying the class cars. A few times the classes truck, bus, and train are falsely classified as cars. This can be explained because these objects also belong to the same categorical group. These are errors that are lower ranked and because of the big similarities between its objects.

By evaluating the model regarding its group IoU a better understanding of the classification behavior can be obtained. This indicates how many critical mistakes the model makes and which groups are difficult to classify. Table 4.18 compares the group IoU for the seven different categorical groups introduced in section 4.1.1. The first line describes the group IoU on the different groups and the second row describes the averaged IoU of the separated classes.
This combination of the evaluation classes pushed performance very much up to 81%. Important is that the performance on the grouped classes is higher than on the separated classes. This means that the classification belongs to the same group and is not completely wrong. In this cases the model has just a classification instead

Flat	Construction	Object	Nature	Sky	Human	Vehicle	Mean
96.04	86.62	55.96	89.19	89.92	70.60	88.71	81.37
87.86	66.01	54.84	74.65	89.92	58.49	72.33	71.99

Table 4.18: Evaluation of group IoU of SCNN

a detection problem because the object is correctly detected but falsely classified. This behavior can be identified for every class which is a very good classification pattern.
The object's class improves just a very little but all other classes rise significantly in IoU. The group sky is built just out of the class sky so it does not change. The object group contains the classes poles, traffic signs, and lights. These are very few objects and often have bigger interference with the background class. This makes the group IoU do not increase a lot.
The other classes increase a lot by grouping them. Especially the human group with the classes persons and riders benefit from this. As well the nature class which is built by vegetation and terrain with just 60%. The confusion matrix has already shown that terrain tends to be classified as vegetation which is why the nature group increases IoU a lot.
Finally also the classification difficulty on motorcycles can be fixed by grouping all vehicles but especially the two-wheelers together. These are often misdetected and now show very good results.

Like explained earlier, especially in section 4.3.1, bigger classes are classified better than classes that occur less in the dataset. This part deals with this relation and compares the IoU with the number of appearances of its class. Figure 4.22 compared these two features over the 19 recommended classes. The figure is constructed with two different scales - the blue one (on the left side) indicates the IoU of the different classes and the red scale (on the right side) shows the number of appearances of pixel from these classes.
The figure shows that there indeed is a correlation between the number of appearances of a class and its IoU. Say every class which appears with at least 2% in the dataset got an IoU of over 80%. These are the classes road, sidewalk, building, vegetation, sky, and car.
The class which appears least in the dataset are motorcycles which as well got the lowest IoU and do not even reach 20% in IoU. The percentage frequency of occurrence for motorcycles is 0.07%.
The other classes state these assumptions without exception although these appearances are very low and close. Their pixel appearances are lower than 2% and the values for its IoU every between 50% and 57%. Other classes like traffic signs, terrain, persons or bicycles occur with around 2% just slightly more often but achieve results of over 60% in IoU.
An exception is the class train which archives with a percentage frequency of occur-

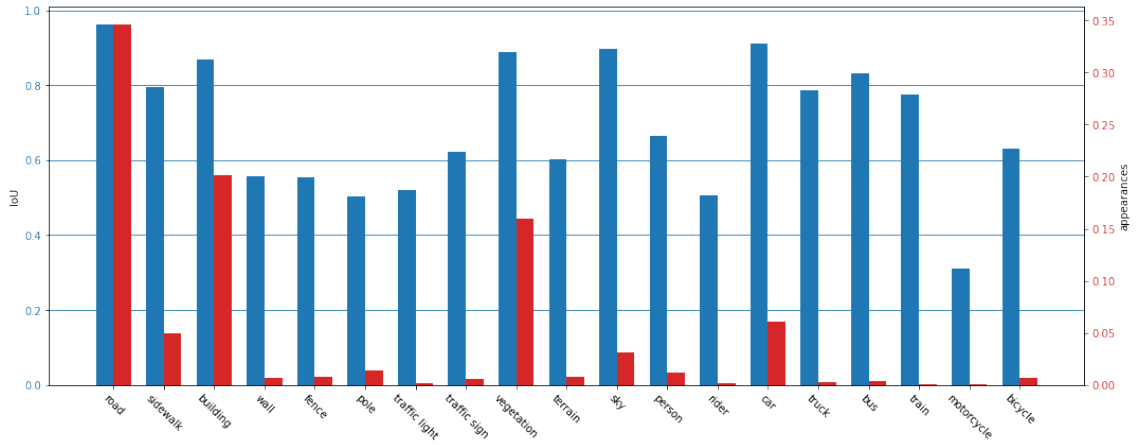


Figure 4.22: Correlation of IoU and number of appearance

rence of just 0.07% very good results.
In general this is a typical pattern for urban street images that classes like the road, cars, and buildings are the classes with by far the most pixel appearances. Include highway images could be a solution to get a higher variance of backgrounds but it cannot solve the problem that there are far fewer trains or motorcycles on the street than for example cars, not to mention background classes like roads, buildings, or vegetation.

This part has shown the influence of the pixel occurrence on its detection and classification performance. Classes that occur with more pixels in the dataset generally get hit better. As shown by the class of persons compared to the class riders, especially in the low occurrence ranges more appearances can improve classification performance significantly.

4.4.2 Compare all models

This part compares the results of the different models (Fast-SCNN, SCNN, ShelfNet, and ConvNet) with each other. It shows the numerical results as well as the visual results of their predictions.

Table 4.19 presents the evaluation of the four models. The models are evaluated regarding their number of parameters, their runtime, their accuracy, their mean IoU, and group IoU.
Comparing the number of parameters the models are very similar but the ShelfNet has more than 6.5 times the parameters of the next biggest model. This is an unusually large number of parameters but is still due to its characteristic shelf structure very efficient.
This gets proven by the runtime of the models. The runtime was calculated by averaging the runtime of 100 predictions with a batch size of 1 and measured in seconds [s]. The experiments are made on a Titan Xp machine with 2 gpus. The

Model	Parameters	Runtime [s]	Accuracy	Mean IoU	Group IoU
Fast SCNN	1.761.284	0.2524	0.9055	66.42	71.99
SCNN	1.883.700	0.2744	0.9137	69.66	81.37
ShelfNet	13.420.063	0.3123	0.9390	73.47	86.14
ConvNet	2.024.526	0.3388	0.9151	63.68	80.58

Table 4.19: Performance evaluation of the presented models

Fast-SCNN is slightly faster than the SCNN and after a little gap the next fastest is the ShelfNet. The ConvNet is the last fastest although it has lots of parameters less than the ShelfNet. Dilated convolutions, like used in the ConvNet slow the model down and are computationally expensive although the parameters do not increase. The accuracy is for the ShelfNet higher than for the other models. The SCNN improves compared to the Fast-SCNN version but still does not surpass the other two models.
In terms of mean IoU the ShelfNet returns as well the far best results. Here the results of the SCNN are better than with the ConvNet although it reaches a higher accuracy. The same goes for the group IoU which is also worst for the ConvNet although the difference to the others got smaller.
To sum things up, the results obtained by the ShelfNet exceed the others in nearly every performance category. This performance can be explained by the far bigger number of parameters of the ShelfNet. The ConvNet reaches good accuracy but does not perform well on the more important IoU metric.

The most important metric for evaluating segmentation tasks is the mean IoU. Diagram 4.23 presents the performance based on its mean IoU on every class separately.

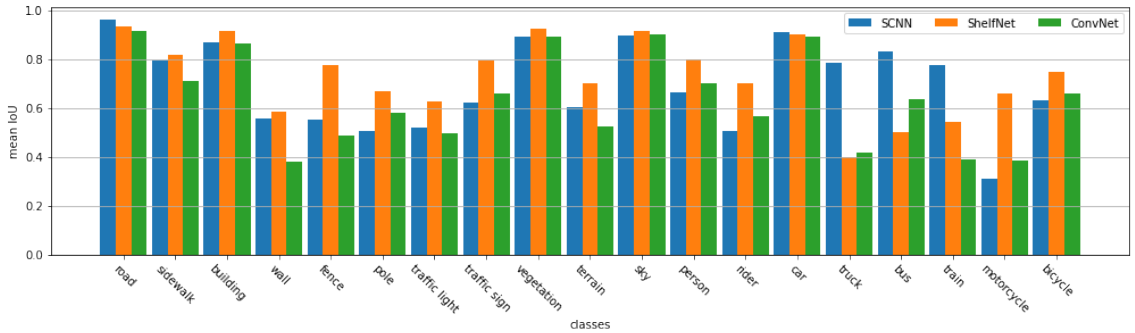


Figure 4.23: Comparison of IoU of SCNN, ShelfNet and ConvNet

The figure shows that the SCNN classifies the vehicle classes very well compared to the other two models. The SCNN reaches far better results on the classes truck, bus, and train which are just hardly classified correctly. On the other hand, the two-wheelers are classified better by the other two models especially with the ShelfNet.

In general it is noticeable that on the bigger classes the performance of the different models is very close. On the other classes, except the three mentioned classes, the ShelfNet performs best. Especially for the classes of the object category (pole, traffic lights, and signs) and on fences the ShelfNet is better.
The performance of the ConvNet is very uneven because it got lots of good classes but as well four classes with a low IoU of around 40%. This happens just very rarely on the other two models.

4.4.3 Reproduce results on new dataset

To generalize the results of the network another dataset is used for evaluation. This section presents the results obtained with the SCNN on a new dataset (mapillary vistas [20]). Therefore the model is trained on this data and evaluated. The results obtained in this section are based on the ones obtained for pretraining in section 4.3.3. Here the model gets trained further to evaluate the performance on this new dataset.

The training is shown in the following figure. As shown in figure 4.24 the model is trained over 200 epochs with the same data augmentation techniques and parameters used as presented in section 4.3.2. Unlike the model trained on the other data, this model is not trained over that many epochs. The significantly bigger number of training samples makes the model converge with fewer epochs and finish its training earlier. The figure shows the progress in accuracy on the left and the progress of the loss value on the left.

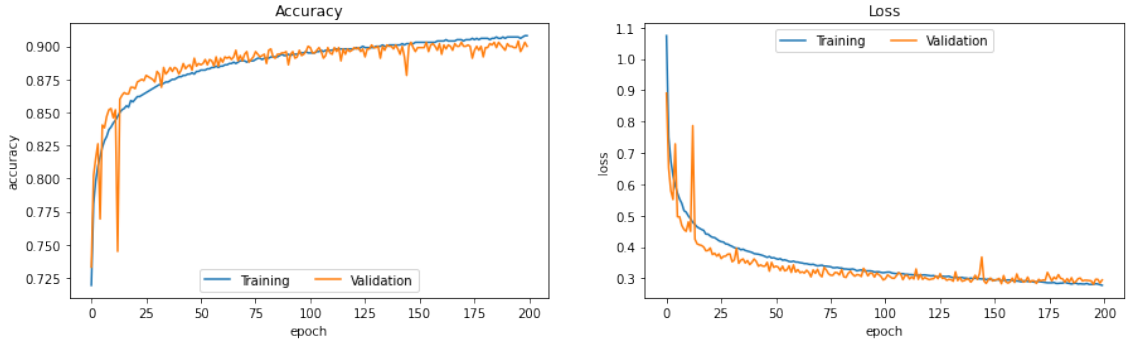


Figure 4.24: Accuracy and loss of training on mapillary data with SCNN over 200 epochs

Like shown in the graphic the accuracy increases monotonously up to 200 epochs. From there on it is not rising any higher. The progress of the loss functions shows the same behavior. First, it is decreasing very fast but from 100 epochs on it is barely falling any further.
The performance on the training and the validation data are very close. From 125 epochs the performance on the training set is surpassing the performance on the

validation data and continues to improve continuously. On the other hand, the performance on the validation data does not improve and more.
In general the progress of these metrics is very smooth and does not contain a lot of outliers where it converges towards a minimum.

Table 4.20 presents the results obtained on this new dataset and compares the results with the ones obtained by training on the cityscapes set. It compares the loss, accuracy, and mean IoU for both data sets.

Model		Mapillary	Cityscapes
Loss	0.2373	0.2895	0.3846
Accuracy	91.89	90.46	91.37
Mean IoU	43.39	41.93	69.66

Table 4.20: Evaluate SCNN trained on mapillary

The results of the mapillary model compared to the ones obtained on the cityscapes dataset are worse on the new dataset. Where the loss value is lower for the mapillary model the accuracy values and most important the mean IoU values are higher for the cityscapes model.
The lower loss value comes from the good classification of the big classes which take up the most space in the images which results in a lower error value. The difference is that the mapillary dataset contains some very big classes which occupy lots of pixels that get classified correctly. On the other classes which occur rarely in the dataset, the performance drops significantly. This gets proven by the good accuracy but the bad mean IoU for the mapillary model.

The difference in mean IoU is shown in figure 4.25 for both models on the dataset they are trained on.

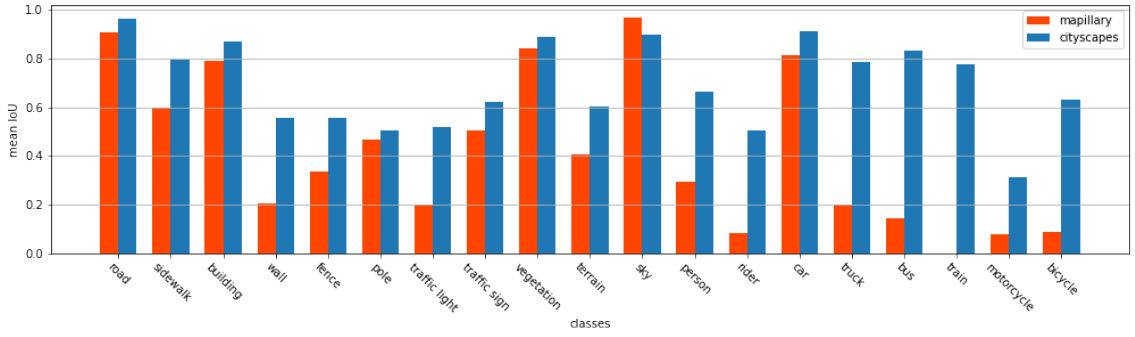


Figure 4.25: Compare IoU on new mapillary dataset

The graphic shows that the performance on the bigger reaches the performance of the cityscapes model. On classes like the road, buildings, traffic signs, vegetation, sky, and cars the performance is close although slightly worse. On the other hand

classes like walls persons, riders, buses or two-wheelers are not classified well.
Like already explained in section 4.3.3 this is due to the frequency of the classes. There are bigger differences because the mapillary data is not exclusively collected inside cities and contains much more variety. E.g trains just occur that little that the model never classifies them.

The following part shows predictions made on four samples of the mapillary data. The first row shows the input images, the second row the ground truth labels, and the last row the predictions made by the system. The samples contain situations inside cities with persons and classes with lots of smaller objects and situations outside the cities where the bigger classes like road, vegetation, and sky dominate the image.

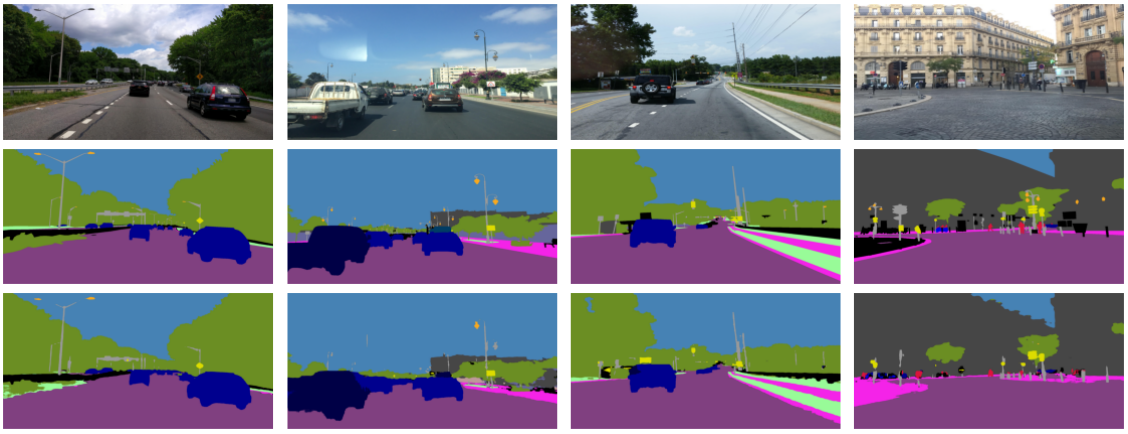


Figure 4.26: Predictions made on the mapillary dataset

The first sample shows a situation on the highway with the bigger classes like road, vegetation, sky, and cars. These classes push the performance of the prediction very much. Moreover, the sample contains lights and poles which as well are detected very well.
The same goes for the second sample. Here as well a truck is represented which is detected well. Its contours at some places seem poorly detected but this is caused by the shadow in the input image. As well the signs lights and fences are detected well. Moreover, some buildings in the background get detected very well.
The third sample shows the same pattern. Especially the different layers next to the road or street signs far away are detected with very high accuracy.
The fourth sample shows a situation inside cities and contains lots of small traffic lights, signs, and fences. Furthermore, this situation contains many static objects which are classified into the rest class. These are hardly classified correctly by the model. The reason, therefore, is that these objects vary very much but do not occur very often.

All in all, these results show that the performance on this different dataset is also very good. This is not only stated by the good accuracy but also by the predictions.

The mean IoU discloses the problems of this new dataset. It contains a very big variety and more different situations than the other dataset but the smaller classes are less represented. These few samples make it hard for the model to learn the small classes.

Chapter 5

Experimental Work - Lane Detection

This chapter describes the experimental part done for the lane detection task. This part applies and evaluates different models to this task. The line detection task is as well treated as a segmentation problem. The lines and background form the two classes for the binary segmentation.
First the different data sets are presented and explained. The next section presents the different models implemented, developed, and evaluated.
Final part of this chapter is the combination of the lane recognition task with the results from the segmentation part. It brings these two parts together and forms a capable perception model for autonomous driving.

5.1 Data Set

This section presents the data sets for the line detection task used in this work. The data sets are available online and are used to evaluate state-of-the-art performances in the fields of autonomous driving. The data sets presented in this work are the TuSimple and the CULane dataset.

5.1.1 TuSimple

This dataset of TuSimple [31] is built out of road images on highways. These images vary in weather conditions (sunny or rainy) and different daytime. The images also vary in terms of traffic conditions which guarantee a great variance of situations. Labeled are up to four different lines to also train the understanding of which line is detected on which position. The green and the blue lines limit the actual line the own car is driving on. The line on the left got marked by a red line and the line on the right by a cyan line.
In general the dataset consists of 3626 annotated images. The labels have to be created out of *.json* files. These polylines get transformed into pixel values to create a label pixel map.
Some images were taken while changing lanes. In this case, one more line is annotated because sometimes it is not clear which lane the actual driving lane is.

The dataset is not divided into training and test sets which has to be done by the user which makes it hard to compare and evaluate results with the state of the art.

5.1.2 CULane

This dataset [30] is far bigger than the dataset shown above. It contains 88880 training samples and 9675 for testing. This makes it easier to apply it to the traffic understanding scenes of the data sets for semantic segmentation. Furthermore, it contains not just images captured on highways but also ones inside cities and different road types. The variance between the images is increased by driving with different cars and drivers. All images are taken in and around Beijing.
Unlike in the other dataset, here are labeled the four most interesting lanes with the same label to not differ which line type it is. Other lane markings like ones on the oncoming lanes or a fifth lane are not annotated.
The lanes are described by cubic splines and later transformed into pixel maps to train on.

5.2 General line detection

This section describes the experimental part of the line detection task. For this section, all lines are treated equally and there is no distinction made between the different lines (on the right/left). This turns this task into a binary classification problem with the two classes - line and background. Such a task can be addressed by using either the binary or the categorical cross-entropy. This part considers both metrics and evaluates them against each other.
This section is structured into training a model on the two data sets separately and then evaluating them in overall experiments using other data to identify the better model.

5.2.1 Training on the TuSimple dataset

The first part describes the training on the TuSimple 5.1.1 dataset. The model used for this task is the SCNN presented in section 4.2.4 of chapter 4. Because of the just two output classes, for this task, there are used the binary and the categorical cross-entropy.

For the binary cross-entropy, the model uses an output layer that returns one class with a sigmoid activation which is either 0 or 1 for background respectively line. On the other hand with the categorical cross-entropy, the output layer with a softmax activation has two output classes.
The results are shown in table 5.1. This table compares the performance of both models in terms of their mean recall, precision, F1 score, IoU, and accuracy.

Metric	DA	Accuracy	Recall	Precision	F1 score	IoU
Binary	-	98.79	80.24	78.73	79.37	70.62
	x	98.86	85.93	77.50	81.45	72.43
Categorical	-	98.80	71.06	83.86	76.83	67.28
	x	98.86	74.76	83.70	78.90	69.78

Table 5.1: Lane detection results on the TuSimple dataset

The table shows that the model trained with the binary cross-entropy returns better results. With the binary cross-entropy, the f1 score reaches more than 80% with data augmentation and an IoU of 72.43%. The accuracy is a very poor metric for this evaluation and does just vary a very little. With both metrics, using data augmentation improves the performance. As data augmentation techniques there are applied the same as presented in the previous chapter in section 4.3.2. The results obtained with the categorical CE are the same in terms of accuracy but in terms of its F1 score and IoU, the binary CE returns better results.
An important characteristic of the binary CE is that it tends to predict thicker lines. This is indicated by the higher recall and lower precision. Higher recall means that it detects more line pixels correctly and lower precision says that it in general predicts the line class more often.

The predictions for one example are presented in figure 5.1. It shows the ground truth label on the left, the prediction of the model with the binary cross-entropy in the center, and the prediction of the categorical cross-entropy on the right.

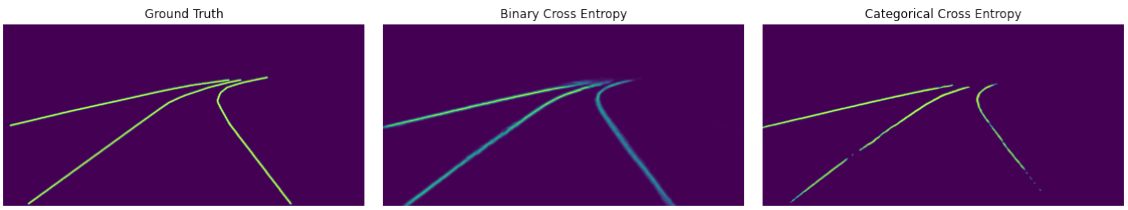


Figure 5.1: Compare predictions on TuSimple data of networks trained with binary and categorical cross-entropy

This shows the classification behavior of both metrics very well. Where the binary case applies an aggressive classification with values in $\mathcal{V} \rightarrow [0, 1]$. It still detects lines very well, although sometimes the output values are not very close to 1. As well, it can be observed that the model predicts thicker lines than they appear in the label. This is the visual explication of the higher recall and lower precision.
On the other hand the categorical case maximizes the probability for one of the output classes and because of this, it just returns values $v \in \{0, 1\}$. Here the predictions are not that thick and similar to the label data. The prediction still contains parts where the lines are not detected and it shows interrupted lines.

The next part analyzes the binary prediction behavior in more detail. Figure 5.2 shows a cutout of a line from the prediction of the previous example for the interval [900:905, 250:330]. Under the cutout image, the figure shows the color bar which indicates the prediction values.

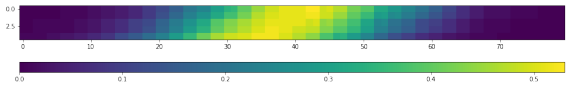


Figure 5.2: Binary prediction of lane detection in detail

This figure proves that the prediction values are higher the closer these are to the ground truth line. The maximum value for this example is slightly bigger than 0.5 and decreases constantly. For other parts the prediction probabilities are higher.

To calculate the mean IoU, precision or recall for this the prediction need to be converted into the space of $v \in \{0, 1\}$. This is made by rounding over a threshold t with the following logic

$$v = \begin{cases} 1 & \text{if } x > t \\ 0 & \text{if } x \leq t \end{cases} \tag{5.1}$$

Table 5.2 compares the performance based on their mean recall, precision, f1 score, and IoU for different thresholds. The goal is to find a threshold that does not permit too many values to be marked as a line (high recall) because this makes the predictions look puffy. To high thresholds cut out important parts of the line and may interrupt them, so it also needs a high precision value. The f1 score combines the recall and precision like explained in chapter Theoretical Background 2.4.

Threshold	0.1	0.2	0.25	0.3	0.35	0.4	0.5
Recall	93.53	90.22	87.36	85.93	83.15	80.44	74.26
Precision	68.96	73.49	75.84	77.50	79.16	80.85	83.79
F1	79.36	80.96	81.11	81.45	81.04	80.58	78.67
IoU	67.02	70.63	71.92	72.43	72.37	72.02	69.54

Table 5.2: Compare performance of IoU depending on threshold value

The table shows that the performance gets better up to a threshold of 0.3. From there on it decreases again. The best performance is obtained with a threshold of 0.3 where both, the IoU and F1 score have a maximum. Especially the negative correlation between recall and precision gets clear. By evaluating the F1 score the best trade-off between both is found.

The predictions including these thresholds are shown in figure 5.3. This figure compares the predictions of four significant thresholds.
It proves that the lines with too low thresholds, like $t = 0.1$ do not look natural. As

Figure 5.3: Binary prediction with different thresholds

well, the predictions with too high thresholds like $t = 0.4$ sometimes cut out parts of the lane. The graphical analysis proves the results of table 5.2, that with thresholds around $t = 0.3$ best results are obtained.

Another technique to improve classification results with an imbalanced dataset like the line detection problem is to apply weights on the loss value depending on the true class. The basics are presented in chapter 2 section Losses and Metrics to assess Segmentation Quality.

The results of training with weighted cross-entropy are shown in table 5.3. This analysis compares the different weight ratios regarding their accuracy, mean IoU, recall, precision, and F1 score. The upper part represents the results of the weighted categorical CE and the lower part the ones of the weighted binary CE.

Weights	Accuracy	Recall	Precision	F1 score	IoU
[1., 1.]	0.9886	0.7476	0.8370	0.7890	0.6978
[.5, 20.]	0.9863	0.8292	0.7186	0.7699	0.6764
[.5, 2.]	0.9863	0.7851	0.7775	0.7807	0.6945
[.4, 1.]	0.9874	0.8182	0.7935	0.8050	0.7186
[1., 1.]	0.9886	0.8593	0.7750	0.8145	0.7243
[.4, 1.]	0.9877	0.7799	0.8034	0.7908	0.7044

Table 5.3: Results of applying class weights for line detection

The table shows that the accuracy is not rising after training the model with weighted class losses but instead it is falling a little. More important is the analysis of recall and precision. The precision is maximum for equal weights and higher than in the other experiments. This indicates that the model tends to predict fewer lines and gains performance by not predicting wrong lines. Applying weights pushes the model to predict more line pixels. This is indicated by the higher recall. Inhomogeneous predictions are indicated by a bigger difference between recall and precision. This mostly ends in parts where the line is detected very well but also other parts where it is not detected at all. This happens by applying to big weights. In general, the best results, argued by high F1 score and IoU, are obtained by applying smaller weights.
Compared to the results obtained by the binary CE the results still do not surpass

the performance trained with the binary CE. The precision values are nearly even but especially the recall is better what leads to the higher F1 score and IoU. Weighting the binary CE loss does not improve results. This leads to a decreased recall but increased precision where the high recall is the advantage of the binary entropy classification.

The next part visualizes the prediction of the different models shown in table 5.3. It compares the prediction of the model trained with CE without using weights and with weights of $[.5, 2.]$ respectively weights of $[.4, 1.]$.

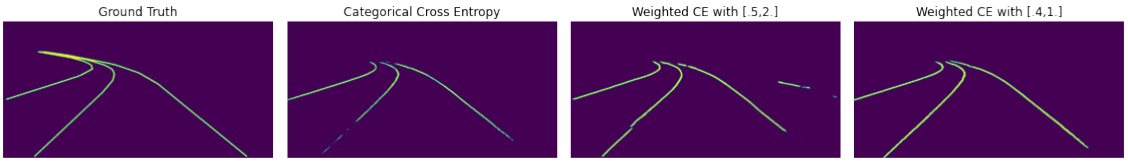


Figure 5.4: Predictions of model trained on weighted categorical cross-entropy regarding their class weights

Here the effect of weighting the class loss is visible. The model without weights does not detect some important parts of the line. The one with the more aggressive weights detects the line better but returns an inhomogeneous prediction. With smaller weights, the detection is more smooth and without any gaps.

After analyzing the correct detection of class 1 due to the recall and precision this part evaluates the accuracy on both classes separately. The confusion matrix evaluates in detail how both classes are classified. Table 5.4 shows the confusion matrix of the models trained with binary CE on the left and the one trained with the categorical CE and weights of $[.4, 1.]$ on the right.

	0	1
0	0.9908	0.0092
1	0.2866	0.7134

(a) Binary cross-entropy

	0	1
0	0.9933	0.0067
1	0.4255	0.5745

(b) Weighted categorical cross-entropy

Table 5.4: Confusion matrices on TuSimple regarding their error metric

The confusion matrix evaluates the pixel-wise accuracy and not the metrics like recall, precision, or IoU. As known, the accuracy of the background class is very high. The error of this class seems very low because of the far bigger number of correctly classified pixels.
More interesting is the performance on class 1. The confusion matrix shows a percentage performance of how many pixels of this class are classified correctly. This shows the big advantage in the performance of the model using the binary CE. This model classifies over 71% pixels of class line correctly, where the model with categorical cross-entropy just classifies 57% correctly.

5.2.2 Training on the CULane dataset

This section describes the results obtained with the CULane dataset of section 5.1.2. This dataset is used to challenge and verify the results obtained before. As introduced in section 5.1.2 the CULane dataset is far bigger and also contains images without street lines as *negative* examples. This makes it more difficult to train on and the training process more extensive.

The results regarding the different error metrics are presented in table 5.5. It is evaluated on the same metrics as the other dataset. The models in this section are trained with data augmentation as well. The table compares the models of both metrics with equal weights and with weights of $[0.4, 1.0]$.

Metric	Accuracy	Recall	Precision	F1	IoU
Binary CE	0.9914	0.6381	0.6817	0.6575	0.6533
Weighted Binary CE	0.9910	0.6006	0.7131	0.6489	0.6525
Categorical CE	0.9910	0.5821	0.7454	0.6493	0.6019
Weighted Categorical CE	0.9901	0.6226	0.6963	0.6545	0.6220

Table 5.5: Lane detection results on the CULane dataset

The results show the same pattern as with the other dataset. The binary model returns better recall but lower precision. In general, it is still better than with the categorical CE. For the binary model, weighting the loss does not improve performance. As well, the values are very close to the performance on the other dataset. As with the previous dataset, the categorical model returns a worse recall but a higher precision. This is not very useful because it detects fewer lines. In general, the accuracy also does not reach the performance of the binary model.

The following figure visualizes the predictions depending on its error metric. The predictions made with the binary model already include the threshold of 0.3.

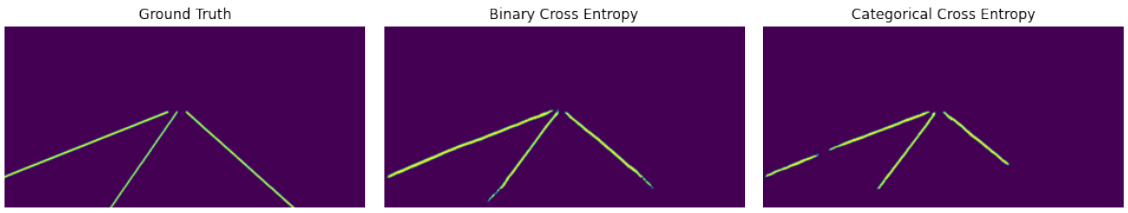


Figure 5.5: Compare predictions on CULane data of networks trained with binary and categorical cross-entropy

As with the other dataset, the binary model returns very good predictions. The detected lines are not interrupted but at the lower parts, the performance decreases a lot. The lower pixels are just very hardly detected. The categorical model shows

the same pattern. Here as well the lower lines are not detected well. In general, the performance of the binary model is better.
Analyzing the dataset it gets clear that the labeling is not very consistent because by changing the lanes the lines are not labeled at all. This context is shown by figure 5.6 which shows a scene while changing the lane where the visible and relevant lines are not labeled.

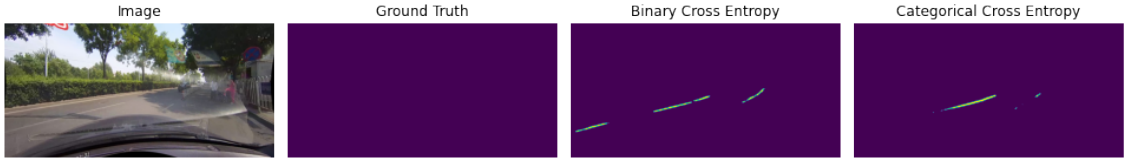


Figure 5.6: CULane labeling and predictions while changing the lane

This figure shows the input image as well as its ground-truth label. In the image, the street lines are visible and for sure relevant but the ego vehicle is located at an angle of 35 degrees. These images are labeled without lines. The predictions show that the model tries to generalize and detect the lines so this inconsistency harms the performance.
The same for the scene shown in the following figure. Here the ego vehicle drives over a car park with one lane as a one-way street.

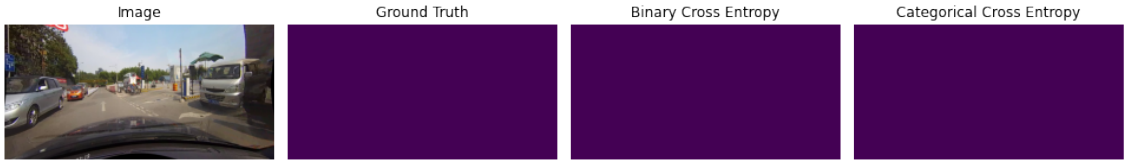


Figure 5.7: Street scenes without line labels

These situations are not labeled at all. The models also do not predict any lines on these situations but this still harms the generalization performance. These examples greatly influence the learning progress during training. Streets without clearly marked lines will not be detected by further models.

The next part analyzes the accuracy of both classes separately and shows the confusion matrix for both models.

	0	1
0	0.9927	0.0073
1	0.3675	0.6325

(a) With binary cross-entropy

	0	1
0	0.9985	0.0015
1	0.523	0.4770

(b) With categorical cross-entropy

Table 5.6: Confusion matrix of predictions on CULane

The confusion matrices show that the background class is better detected with both models. Still, the binary model classification contains lots of FP. The value is high compared to the big number of pixels in this class. A reason for this is examples like the one shown in figure 5.6 where the line is not labeled but the model still tries to detect it. The classification accuracy of class 1 is lower what indicates the difficulties of detecting the lines properly.
The categorical model is better at not detecting missing line labels (FP) but also detects less often the correct lines (TP). The poor line detection ability is already indicated by the lower recall. In general, the binary model is better than the categorical one. Especially if it is considered that some FPs are logically correct but labeled inconsistently.

5.3 Detailed lane detection

This section treats the topic in a special way where every detected line gets further classified into which lane this line belongs. This creates a multiclass classification of the foreground class into up to four different lines. These are the two lines defining the ego lane and the one on the right and the left. Often there are detected less because the street does not have that many lanes.

In general this problem is tackled the same as in the experiments described before. It uses the same model and just changes the classifier in the output layer. Here it returns 5 output classes and just uses the categorical cross-entropy.
Table 5.7 presents the results of the experiments made for this task. The experiments also use data augmentation but do not mirror the images because this would confuse the labeled lines. The different experiments are evaluated on their accuracy, mean IoU, recall precision, and F1 score.
The upper part of the table evaluates the effect of using data augmentation and the other experiments on how weighted classes benefit training.

Weights	Accuracy	Recall	Precision	F1	IoU
[1., 1.]	0.9866	0.4401	0.7305	0.5455	0.4262
[1., 1.]	0.9874	0.4683	0.8008	0.5828	0.4465
[.4, 1.]	0.9856	0.6324	0.6416	0.6346	0.5191
[.5, 2.]	0.9855	0.6241	0.6343	0.6267	0.5011
[.5, 20.]	0.9819	0.6960	0.5891	0.6318	0.4951

Table 5.7: Results of applying class weights for detailed line detection

From the first two lines of the table, the benefit from data augmentation can be observed. The first experiment shows the results without and the second on the results with data augmentation. This leads to a significant increase in both metrics - F1 score and IoU. It can be observed that both the recall as well as the precision

increase significantly. This means that the model, not just only predicts the line class more often but it hits the line class better. This is indicated by a noticeable higher F1 score.

The predictions for these two cases are shown for one example in figure 5.8. The first image shows the ground-truth label, the second one the prediction of the model trained without, and the third on trained with data augmentation.

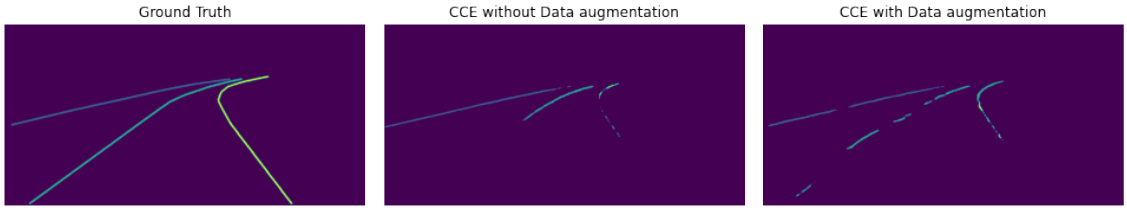


Figure 5.8: Compare predictions of categorical cross-entropy and training with data augmentation

Figure 5.8 shows that the model faces big problems detecting the lines properly. First of all both models are not able to detect the complete lines solidly. The predictions contain a bigger part where the line is not detected. The model trained with data augmentation is better but the performance is still not sufficient. Furthermore, it cannot determine the exact labels of which line is detected and the predictions show multiple lines with the same label. This is observed in the figure on the right image, where the second and third lines are assigned the same label (color).

As indicated in table 5.7, weighting the classes leads to better performance. The table compares the three different weight ratios $[.5, 20.]$, $[.5, 2.]$ and $[.4, 1.]$ where the lowest weights return best results. As observed in experiments in the sections before, this states that too high weights make the model predict the line class to aggressive. This is indicated for the model with bigger weights by the high recall and lower precision. This means that it predicts the line class more often correct but also contains more wrong predictions on this class. The combined F1 score is still high, but there is a big difference between recall and precision. Lower weights lead to smoother training and better predictions. Especially the weights $[.4, 1.]$ return very good results which surpass the others clearly in terms of F1 score and IoU.

This is proven by their predictions presented in figure 5.9. This figure compares the predictions of the models trained with these three weight ratios.
Like argued with the numerical results the predictions of the models with more aggressive weights ($[.5, 2.]$) are worse because they tend to predict too thick lines which contain lots of FP around the lines. Worse, that these models also often contain parts where the lines are not detected at all. Generally, a bigger difference between recall and precision leads to these results.
The models with smaller weights treat this problem way better and do just very

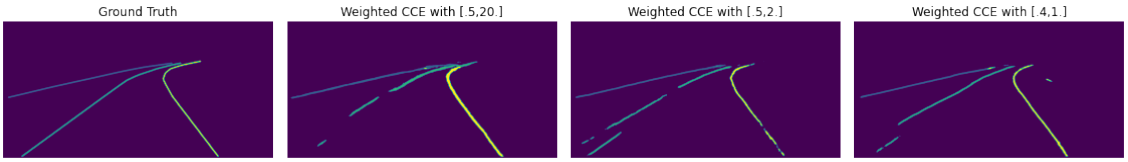


Figure 5.9: Compare predictions of categorical cross-entropy depending on weights

rarely contain interrupted line predictions. As well, their predictions have a better precision which leads to more detailed predictions. The capacity of identifying the exact label of the detected line is equal for both models.

The following will evaluate the performance of every class separately by analyzing the confusion matrix. This is shown in table 5.8. The confusion matrix presents the results obtained with the model trained on the class weights $[.4, 1.]$. It shows the results normalized over the true classes.

		Predictions				
		0	1	2	3	4
Labels	0	0.994	0.002	0.002	0.002	0.
	1	0.372	0.568	0.038	0.014	0.008
	2	0.496	0.012	0.424	0.056	0.012
	3	0.454	0.	0.010	0.520	0.016
	4	0.610	0.	0.	0.158	0.228

Table 5.8: Confusion matrix with detailed lane detection

The first thing to observe from this representation is the big difference in classification accuracy between the lines and the background class. The accuracy of this class is that high because of the big imbalance and the majority of the pixels are always assigned to the background class.
This is also the biggest problem of the model in that it tends to classify line pixels as background. This is indicated by the high percentages of the predictions for class 0. This means that the difficulty of this task is the detection and not the classification into the correct label. As shown in the confusion matrix, the errors in mixing up the label for the different lines are very small.

Summed up, the detailed line detection does not lose accuracy compared to the model with the general line detection. The difference in the error metrics is that the averaging over more classes with lower accuracy reduces the overall accuracy value significantly. The predictions have shown that there is no need to combine the different lines to one line class.

5.4 Cross evaluation of line detection

This section evaluates the results obtained for the line detection model by running it on the cityscapes data. By including a different dataset an impression of the overall performance is obtained. First, the results of the different models are compared among each other. Later the results are combined with the ones of the segmentation task to obtain an overall model capable of doing both tasks.

5.4.1 Applying the line detection network on the cityscapes data

Here the different models are applied to the cityscapes data and are compared with each other. This contains the models trained on the TuSimple and CULane data and finally the model trained on the detailed line detection.

TuSimple model on cityscapes data

First, the performance of the models trained on the TuSimple data is analyzed. Therefore as the best models of table 5.3 the binary model with equal weights and the model with categorical cross-entropy with the weights of $[.4, 1.]$ are chosen. This also compares the models trained on the binary and categorical CE with each other. Figure 5.10 presents the performance of both models on four examples. The examples include images with straight and curved lines, streets with and without road markings. The first row shows the predictions of the binary model and the second one the predictions of the categorical model.

Figure 5.10: Line detection trained on TuSimple applied to cityscapes data

The first example represents a straight road where on the right side the road markings are missing. The binary model detects both lines but the road limitation without a line on the right side is not detected at all. The categorical model just detects the line in the middle of the road. As well it contains a few false predictions evoked by traffic signs on the road.
The second example shows a straight road with visible lines on both sides. Both models detect them very well and without interruptions.
In the third example, there is just one line on the left which is detected by both

models very well. On the right side where the road is limited by high curbs, the models do not detect the end of the lane because the lines are missing.
Example four shows a curved road where both models detect just the central line but not the limits of the street because these are not marked with a line but with curbs.
Summing up, these models detect the lines very well on a different dataset and in different situations. The big problem is that it does not detect road borders without any markings. This happens in an urban environment very often but the TuSimple data is just trained on highway data.

CULane model on cityscapes data

Next, the models trained on the CULane data are used on the cityscapes dataset. The performance on the same four examples as in the part before is presented in figure 5.11. As in the part before the first row shows the predictions of the best binary model and the second one the predictions with a categorical model (table 5.5).

Figure 5.11: Line detection trained on CULane applied to cityscapes data

The results show a few characteristics of the predictions with these models. First of all the models have problems detecting the lines in the lower part of the input image. These parts fail very often whereas the higher parts are detected way better. This can be observed in all of these four examples and as well on the CULane dataset.
Furthermore the detected lines often are interrupted and detected very sloppy. For instance, the models have bigger problems to detect curved roads like the one in the last example. This might result from the inconsistent labeling in the CULane dataset. But as well straight lines like the ones in the second example are not detected perfectly.
On the other hand this model can detect the end of the road indicated by curbs. This model is not just looking for street lines but also for lane limitations. The first and the third example show this very clear. Here the right border is detected without any line. Compared to the other models in figure 5.10, this is a big advantage.
In summary, these models trained on the CULane data perform worse on the cityscapes data than the ones trained on the TuSimple set. Although the big size of this dataset is not able to generalize well for new data. The strong points of these models are the detection of curbs without line-markings.

Detailed line detection model on cityscapes data

As well the model to detect the exact lines is evaluated on the cityscapes dataset. The advantage of detailed line detection is that it returns more information about the context of the detected line. Knowing which line is detected enables more space for further systems like lane detection functions.

In general this model has the same detection pattern and same problems as the model trained on the general line detection. Both are trained on the same data which makes the output performance of both models very similar. Problems which this model also faces, are especially street borders without lines (like shown in figure 5.10). On roads with existing lines, the model works very well but often it happens that this is not given.

Figure 5.12 shows two different scenes with the predictions made on them. The first column shows the normal line detection function where the model marks the detected lines with their corresponding colors. The middle column filters the two lines limiting the ego lane by filtering these two classes. The final column highlights the ego lane by marking the space between these two lines.

Figure 5.12: From line detection to ego lane detection

The line detection results show that the model detects all of the three lines very well. The lines are completely detected and as well, every line is assigned the correct label. Only weak point is that the marking of the parking slot on the sidewalk is marked as another line. As well this line is assigned the wrong class, actually the class label of the line on the left next to the ego lane. These two images represent the performance on straight roads where lines are visible.
The middle column filters the two lines limiting the ego lane. This works well because the lines are classified with unique labels and the predictions do not mix up classes on the same line.
After filtering the two lines of the ego lane the space between them can be calculated. For these streets with visible lines, the lines are detected without interruptions. In the first example, the model classifies a few parts of the Mercedes star as a line

which interrupts the lane marking for a few pixels.
In general the detailed line detection works as well as the general lane detection and does not lose performance due to the higher number of classification classes and the bigger imbalance resulting from this.

5.4.2 Joining line detection and scene segmentation

This section presents possibilities to combine the results obtained in chapter 4 to classify all objects with the newly developed line detection model. Especially the combination with street detection enables better detection of the lanes.
In the following figure, there are presented four examples of including the line predictions into the segmentation task. The line predictions are colored yellow.

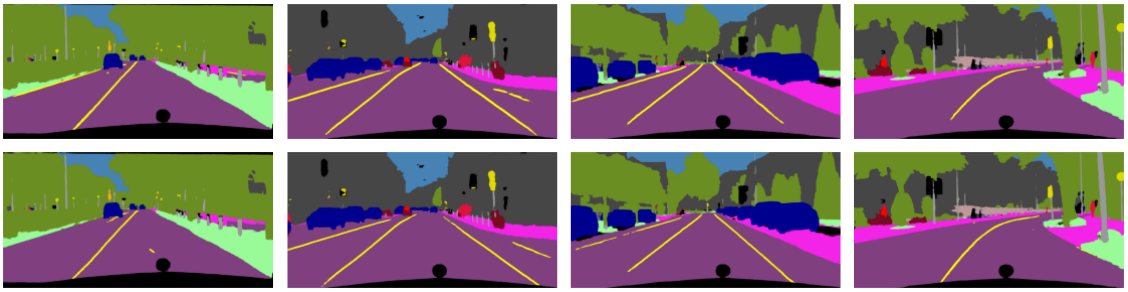


Figure 5.13: Including the line predictions into the segmentation task

The images show how much the detected lines improve the scene understanding. The knowledge of the road increases a lot by monitoring the lines on it. There it does not matter that the road limiting lanes are not always detected because this is already indicated by the road labels. This can be observed very well in the examples one and four in figure 5.13. The road is divided into two lanes by the one detected line.
The other two examples represent situations where the street contains more than just two lanes. Here all important lines are detected to mark the different lanes using the information of the segmentation and the line detection tasks.

This combination of road and line detection makes it possible to detect the different lanes. The following figure 5.14 presents an ego lane detection by combining these two tasks.
Using the detailed line detection model presented in figure 5.12 to identify the class of which line is detected, it can be determined both or one of the two lines limiting the ego lane. For the two lines 1 and 2, the space in-between defines the ego lane like shown in figure 5.12. This is shown in the second and third example of figure 5.14.
If just one line is detected, this mostly will be the line with label 1 which means the left line limiting the ego lane. Without a line with label 2, it can be assumed that there is no lane on the right and the lane is limited by the end of the road. As

Figure 5.14: Using line detection and segmentation results for lane detection

shown in examples 1 and 4 in figure 5.14 this unambiguously identifies the ego lane. It can also happen that neither line 1 or line 2 are detected but line 3. This indicates a situation with at least two lanes but the ego lane cannot be determined. This situation is described by figure 5.15.

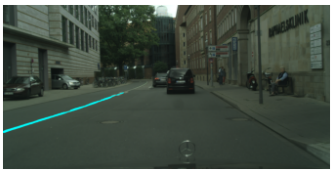


Figure 5.15: Lane detection not possible due to insufficient street lines

Here the road is too wide but no line is defining both lanes. There is just one line separating the next lane on the left for oncoming traffic. This type of situation cannot be solved by the information gathered from these two models.

To find a conclusion the models got their characteristics and although the models trained on CULane can determine the end of the road without markings and the models trained on the TuSimple data cannot, the models trained on the TuSimple data prove to be better suited applying them on new data sets. The models trained on the CULane data have problems with detecting clear lines in the lower part of the image and are not labeled unique which leads to weaker prediction results.
On the other hand the TuSimple models turn out to work very well also inside of cities, although they are just trained on highway data. The weak point is that these models require visible street lines to detect the lines and do not detect curbs. Between the binary and categorical models, there is no better type and both perform similarly.
As well the model to detect the exact lines works very well. This gives much more room for improvements and further functions because it returns very detailed information about the whole scene. The evaluation shows that the accuracy does not drop significantly by splitting the lines up into different labels.
Especially the combination with the segmentation model the missing capacity of not detecting curbs is covered up by detecting the whole road and like that identifying the different lanes. But like figure 5.15 shows, there are still situations that cannot be handled by these models.

Chapter 6

Discussion of Results

This chapter reviews the results obtained in the two chapters before. It deals with the segmentation and the line detection task one after the other. The results will be summarized, put into context and an interpretation will be given. Finally, the limitations are disclosed and future potential is revealed.

6.1 Scene Segmentation

The results obtained in the semantic segmentation task reach state-of-the-art performance. Therefore different models are implemented and evaluated on the cityscapes data. As shown in table 4.19, the performance of all models is close. Best results return the ShelfNet but on the other hand, its number of parameters and runtime is far higher. Although it still belongs to real-time networks. Out of the presented models, the ConvNet is the one with the lowest accuracy and also the least efficient model. The Fast-SCNN also tackles state-of-the-art performance but faces classification problems with smaller classes. By changing the network structure a little bit to extend the upsampling part, the network performance improves. Especially performance on the smaller objects improves. Due to the more extensive upsampling, the runtime also increases a little bit. On the other hand, it still is far faster than the other two models.

Due to the limited dataset of just 3000 examples the training of these models requires data augmentation. The results presented in table **??** indicate that more aggressive techniques like shifting, zooming, and flipping greatly effects training. Pixel varying techniques like brightness changes, applying noise, and channel shifts need to be combined with other more aggressive techniques to lead to bigger improvements. By combining them and increasing the training volume the results can be improved significantly.
Furthermore pretraining influences the performance less. Pretraining on another segmentation dataset like the mapillary vistas data, increases the performance just a little bit. On the other hand pretraining on classification data sets like the ImageNet does not improve performance [9] at all.

Very important knowledge returns the experiments with different image resolutions. These state that the performance rapidly drops for smaller image resolutions. More important the mean IoU suffers a lot because especially smaller classes drop lots of classification accuracy. These results are presented in table 4.17 which compares the original resolution with a half and a quarter resolution.
The publisher of the dataset recommend grouping smaller classes into a class of rest labels. Training on all classes leads to a slightly dropping accuracy. The difference gets obvious analyzing the IoU like shown in table **??**. By training on all classes the smaller classes are just very weakly detected. This leads to a big drop in IoU. But that are not just the grouped classes that show these poor classification results. As well, other classes that occur less in the dataset return low IoUs. With the grouped training classes the performance on these classes rises partly drastically like shown in figure **??**.

Table 6.1 compares the obtained results to the state of the art. It shows the three models with their performances indicated in their respective publications and the implementations in this work. The upper part of the table shows the performance of the models from their publications and the lower part compares the results of the implementations from this work.

	Class IoU	Group IoU	fps (1024x2048)
ShelfNet 2019 [11]	74.8	-	36.9
Fast-SCNN 2019 [9]	68.0	84.7	62.1
ConvNet 2017 [7]	69.15	86.5	11.4 (1920x1280)
ShelfNet	73.47	86.14	
Fast-SCNN	66.42	78.39	
SCNN	69.66	81.37	
ConvNet	63.68	80.58	

Table 6.1: Comparison of class and category IoU, number of parameters and frames per second (fps) to relevant state of the art results for real time semantic segmentation

The results show that the original models are still a little better than the here presented implementations. The biggest difference in performance appears in the implementations of the ConvNet. The ShelfNet reaches the original performance very close. The Fast-SCNN as well drops a little performance to the original models. The redesigned SCNN surpasses the original accuracy of the Fast-SCNN.
Due to the same models and dataset this results from different training parameters. The models here are trained using the adam optimization which adapts the learning rate and momentum on its own, while training. Self-adjusting the learning rate depending on the training progress is a promising approach to increase training results. Using the Stochasitc Gradient Descent (SGD) is another option to manage the training progress. The SGD often is locally more unstable and more likely to

converge to the minima at asymmetric valleys, which often have better generalization performance. On the other hand, the adam optimizer works great with the default parameters and is significantly faster.
On the other hand the models in this work are trained on a batch size of just 4. These are very few examples and the training progress suffers from this too. The other models are trained on batches of 16 or 32 examples. This leads to weaker results as well. Promising would be to implement a weighted loss function to pay more attention to the smaller classes to push these and to balance the classification performance.
The implementation of the network is similar to the ones of the publications and as well as data augmentation techniques there are chosen the same ones. Differences could exist in parameterizing the data augmentation techniques which also can lead to different results.

The best model obtained in this work is the custom SCNN. This model presents the best combination of runtime and performance. The evaluation through the confusion matrix in figure **??** shows that the model still has problems in detecting smaller objects properly. Separating them from the background causes lots of True Negatives, where the model predicts a background class (mainly building or vegetation) instead of for example traffic lights. As well, there occur still often correlations between similar classes like riders and pedestrians or terrain and vegetation. By analyzing the group IoU like done in table 4.18, lots of classification errors disappear due to classifications into false classes which still belong to the same groups.

As well it is shown that this model can be transferred to another dataset to reach good results. The performance is strongly connected to the dataset and the detection pattern changes a lot. Inner-city objects are better detected when training on the cityscapes data. On these classes, the model trained on the new dataset lacks performance. This states the fact that the cityscapes dataset is used for state-of-the-art performance.

Nowadays, in autonomous systems, most object detection systems consider the detection as a Bounding Box regression problem. However, this compact representation is not sufficient to explore all the information of the objects. Even if some objects like bicycles are detected somewhat blurred by the segmentation, this model still provides more information than an ordinary model via bounding boxes.
Next steps would have been to extend the model to an instance level to separate all objects of the same class for further information. These need to be done especially for the foreground classes like persons, vehicles, and traffic lights/signs. This step is crucial to return this information to fuse it with the information of other sensors and then base the driving functions on them.

6.2 Line Detection

The line detection task compares three different models trained on different data sets and with different error functions. Therefore it uses the TuSimple and the CULane dataset. The models trained on the TuSimple data return results up to an F1 score of 81%. These results are obtained with the binary cross-entropy. The best model using the categorical cross-entropy reaches 80%. These models use a weighted error training where the binary model is trained without weights. The prediction patterns are quite different. The binary model reaches a very good recall score where the categorical models reach better results on the precision score.
The models trained on the CULane data have similar characteristics. Also with this data, the binary model turns out to be better due to the higher recall score. Still the results cannot reach the performance obtained with the model trained on the TuSimple data. With the CULane set, the models reach just 65% with both the binary and the categorical model. Weighted training improves results just on the categorical model.
By training on the exact line labels, the performance reaches similar results as with the grouped line labels. Here the F1 score reaches 63%. The lower value comes from averaging over the background and the four foreground classes instead of just over the one back- and one foreground classes.

The goal of the work is to transfer the line detection task to the cityscapes dataset. These two training sets show lots of bigger differences like the TuSimple dataset contains highway data from the United States of America, the CULane data contains data from the area around Beijing with inconsistent labeled data, and the cityscapes set contains images from German cities. These make it difficult to apply models, trained with one set, to the other data. Still, the models perform very well, especially the model trained on the TuSimple data. The TuSimple data leads to a prediction pattern that the model just detects lines but ignores curbs as road limitations. Although the line detection still works very stable on the cityscapes dataset. Otherwise the CULane data contains images from an urban environment and its models can detect curbs as road limitations. But the confusing labeling in this dataset leads to clearly worse results. This dataset contains images from lots of different scenes in daylight, at night, while raining, etc. but the labeling does not serve to apply it to the cityscapes data.
Especially the model to detect the detailed lines turns out to be very useful for further tasks. On its own, it is just able to detect the lines on the street what does not serve very much to the understanding of traffic scenes. But in combination with the segmentation task, especially with road detection, the ego driving lane can be defined. This makes this model very powerful. This detection is not possible with the general line detection because it has to be defined which line is detected. When the road limiting lines are not detected, the model has to return more details about the lines that are detected. This model serves very well for applying it to the cityscapes dataset. Though it just detects the lines, it works very well on unknown

data.

The best models are the binary and the weighted categorical model trained on the TuSimple dataset. The categorical model with weighted classes returns similar results for the detailed and general line detection. Because of this, the model of the detailed line detection will be selected as the best one. Comparing it with the binary model it is more susceptible to False Positives. Sometimes it happens that it classifies pixels of the street or more often the Mercedes star as street lines. As well, sometimes the system reacts to arrows that indicate lane changes on the road. These are FPs as well.
The more important limitation of these models is that they do not detect the road borders. If these are without lines the system cannot detect them. This is mainly due to the training on the highway images. On the highway, the road lines always are uniform and complete. Regarding this point, both systems perform similarly.

Chapter 7

Conclusion

The perception systems are crucial for the performance of SDC. Therefore it is important to gather as much relevant information as possible but still meet the requirements in terms of efficiency and runtime of the embedded systems.

This research has shown that more complex architectures like the ShelfNet clearly surpass basic encoder-decoder networks like the ConvNet. Although these come with more parameters these are not significantly slower but better in performance. Otherwise different architectures like the Fast-SCNN prove that by using more efficient convolutional layers the number of parameters decreases a lot without losing performance.
As well the classification pattern got clear, between which classes occur classification errors like pedestrians, riders, or different types of vehicles and which classes are just hard to detect like traffic lights or poles. This is very important because especially these objects are very important to the goal of driving autonomously.
The classification problems can be corrected by grouping the different classes appropriately into bigger categories. This still returns the needed information about the situation, regarding where, is the driving space, which area is separated, which static or dynamic obstacles appear in the scene, and which traffic participants are there. As well this research shows in detail how the performance would suffer by classifying the objects even more detailed. This is an important factor for performance as well as image resolution. The image resolution weakens performance very much why it is that important to also develop better cameras to ensure the best quality while performing in real-time.
The lane detection part adds sufficient information to the scene detection to detect the road. Although the used data sets do not fit the segmentation dataset perfectly, the results of the combination of the road and street line detection are very good. Especially this part already proposed an instance-level detection of the lanes to differ between the detected lines.
The use of multiple very different data sets and the overall good performance states that the developed model is very appropriate for these tasks of image segmentation in real-time.

This will be the most important part to apply to object detection too. For autonomous vehicles, it is important to distinguish between different objects of the same class to estimate their exact behavior. This just can be done by detecting persons, vehicles, or traffic signs on an instance level. Furthermore, more complex loss functions can be used for training to push performance on the smaller classes. The effect of this weighted loss is presented in the line detection part and helps to improve detection performance.

Now it is shown, that the state-of-the-art performance in perception systems already reaches or even surpasses the human eyes. By designing algorithms that make decisions based on these data the traffic safety can be improved a lot. Especially because these systems always decide by taking all information into account. Eliminating human error in driving situations is a great impact and will change traffic safety for good. By this work, it is shown what already can be realized and how much the development depends on the provided data.

Bibliography

[1] H. Waschl, I. Kolmanovsky, F. Willems, *Control Strategies for Advanced Driver Assistance Systems and Autonomous Driving Functions*, Sringer, 2019, ISBN 978-3-319-91569-2

[2] J. Gamba *Radar Signal Processing for Autonomous Driving*, Signals and Communication Technology, Springer, 2020, ISBN 978-981-13-9193-4

[3] S. Chakraborty, J. Eberspächer, *Advances in Real-Time Systems*, Springer, 2012, ISBN 978-3-642-24348-6

[4] U. Michelucci, *Advanced Applied Deep Learning*, Convolutional Neural Networks and Object Detection, Apress, 2019, ISBN 978-1-4842-4976-5

[5] M. Rezaei, R. Klette, *Computer Vision for Driver Assistance*, Simultaneous Traffic and Driver Monitoring, Computational Imaging and Vision, Spinger, 2017, ISBN 978-3-319-50551-0

[6] R. Klette, *Concise Computer Vision*, An Introduction into Theory and Algorithms, Undergraduate Topics in Computer Science, Springer, 2014, ISBN 978-1-4471-6320-6

[7] E. Romera, J. M. Álvarez, L. M. Bergasa and R. Arroyo, *Efficient ConvNet for real-time semantic segmentation*, 2017 IEEE Intelligent Vehicles Symposium (IV), Los Angeles, CA, 2017, pp. 1789-1794, doi: 10.1109/IVS.2017.7995966.

[8] R. P. K. Poudel, U. Bonde, S. Liwicki, and C. Zach, (2018), *ContextNet: Exploring Context and Detail for Semantic Segmentation in Real-time*, Toshiba Research, Cambridge, UK.

[9] R. P. K. Poudel, S. Liwicki, R. Cipolla, *Fast-SCNN: Fast Semantic Segmentation Network*, 2019, Toshiba Research, Cambridge University, UK.

[10] K. Rimal, *Fast-SCNN explained and implemented using Tensorflow 2.0*, May 7, 2019, `https://medium.com/deep-learning-journals/fast-scnn-explained-and-implemented-using-tensorflow-2-0-6bd17c17a49e`, `https://github.com/kshitizrimal/Fast-SCNN`

[11] J. Zhuang, J. Yang, L. Gu and N. Dvornek, *ShelfNet for Fast Semantic Segmentation*, 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW), Seoul, Korea (South), 2019, pp. 847-856, doi: 10.1109/ICCVW.2019.00113.

[12] J. Long, E. Shelhamer and T. Darrell, *Fully convolutional networks for semantic segmentation*, 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, 2015, pp. 3431-3440, doi: 10.1109/CVPR.2015.7298965.

[13] Ronneberger O., Fischer P., Brox T. (2015) *U-Net: Convolutional Networks for Biomedical Image Segmentation*, In: Navab N., Hornegger J., Wells W., Frangi A. (eds) Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015. MICCAI 2015. Lecture Notes in Computer Science, vol 9351. Springer, Cham.

[14] Paszke, A., Chaurasia, A., Kim, S., Culurciello, E. (2016). *ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation* ArXiv, abs/1606.02147.

[15] H. Zhao, X. Qi, X. Shen, J. Shi, and J. Jia, *ICNet for Real-Time Semantic Segmentation on High-Resolution Images* In ECCV, 2018.

[16] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, *Pyramid Scene Parsing Network*, In CVPR, 2017.

[17] D. Mazzini, *Guided Upsampling Network for Real-Time Semantic Segmentation*, In BMVC, 2018

[18] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L. Chen, *MobileNetV2: Inverted Residuals and Linear Bottlenecks*, 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, 2018, pp. 4510-4520, doi: 10.1109/CVPR.2018.00474.

[19] 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*, in Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[20] G. Neuhold, T. Ollmann, S. R. Bulò and P. Kontschieder, *The Mapillary Vistas Dataset for Semantic Understanding of Street Scenes*, 2017 IEEE International Conference on Computer Vision (ICCV), Venice, 2017, pp. 5000-5009, doi: 10.1109/ICCV.2017.534.

[21] V. Badrinarayanan, A. Kendall, and R. Cipolla, *SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation*, TPAMI, 2017.

[22] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy and A. L. Yuille, *DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs* arXiv:1606.00915, 2016.

[23] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, K. Keutzer *SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size*, In ICLR, 2016

[24] G. Nanfack, A. Elhassouny, R. O. H. Thami, *Squeeze-SegNet: A new fast Deep Convolutional Neural Network for Semantic Segmentation*, The 10th International Conference on Machine Vision (ICMV 2017), 2017

[25] X. Pan, J. Shi, P. Luo, X. Wang, X. Tang, *Spatial As Deep: Spatial CNN for Traffic Scene Understanding*, AAAI 2018, 2017

[26] Kim J., Lee M., *Robust Lane Detection Based On Convolutional Neural Network and Random Sample Consensus*, Neural Information Processing. ICONIP 2014, Springer

[27] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, Y. LeCun, *Overfeat: integrated recognition, localization and detection using convolutional networks*, international conference on learning representations, 2013.

[28] P.-.R. Chen, S.-.Y. Lo, H.-.M. Hang, S.-.W. Chan, J.-.J. Lin, *Efficient road lane marking detection with deep learning*, 2018, IEEE 23rd International Conference on Digital Signal Processing (DSP), IEEE

[29] Zou Q., Jiang H., Dai Q., Yue Y., Chen L., Wang Q., *Robust lane detection from continuous driving scenes using deep neural networks*, IEEE Transactions on Vehicular Technology., 2019.

[30] X. Pan, J. Shi, P. Luo, X. W., X. Tang, *Spatial As Deep: Spatial CNN for Traffic Scene Understanding*, AAAI Conference on Artificial Intelligence, 2018

[31] tusimple-benchmark, *TuSimple Lane Detection Challenge - Training Dataset*, CVPR 2017, 2017

[32] Yazeed Alaudah, Patrycja Michalowicz, Motaz Alfarraj, Ghassan AlRegib, *A Machine Learning Benchmark for Facies Classification*, arXiv:1901.07659, 2019

[33] Motor-bw, Assistenzsysteme von Bosch – mit Bosch sicher und entspannt ans Ziel: ACC und ACC Stop & Go, 2021