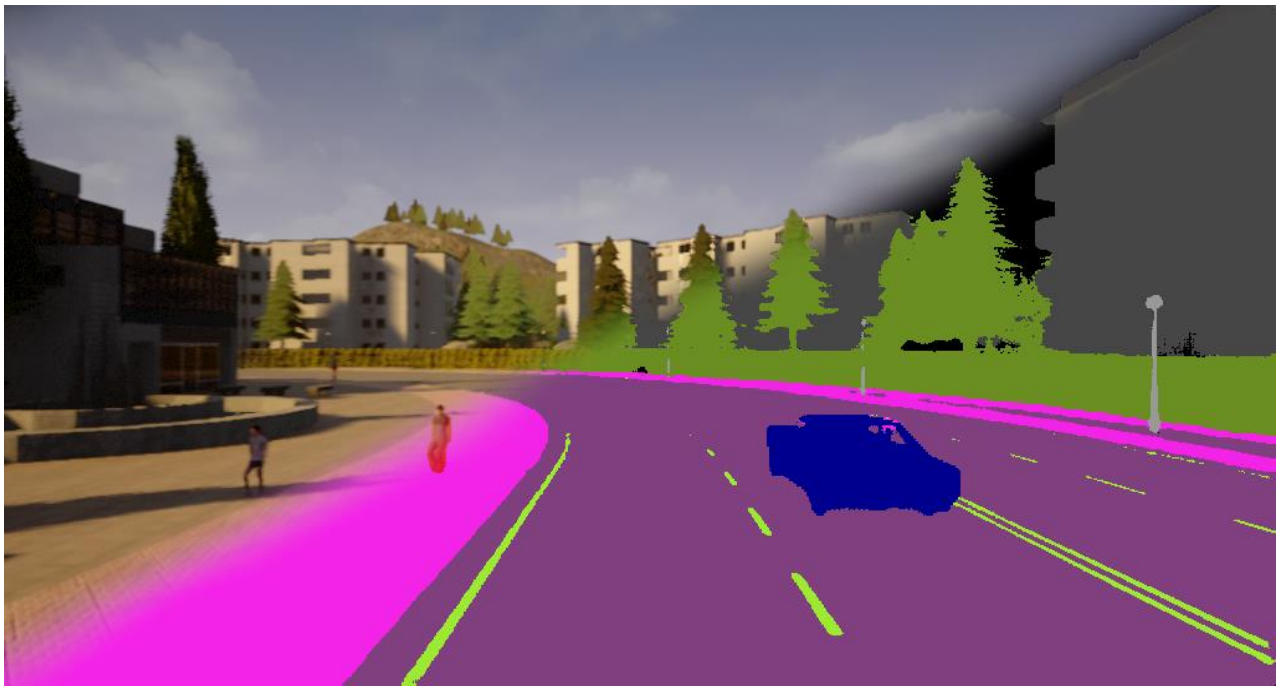
CHALMERS
UNIVERSITY OF TECHNOLOGY



Segmented Classification of Traffic Environments Using RGB-D Data

Considering the effect of image resolution and the relevance of artificial data during training

Master's thesis in Systems Control and Mechatronics & Engineering Cybernetics

JOHANNES LINDGREN
FLORIAN ODEHNAL

DEPARTMENT OF MECHANICS
AND MARITIME SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020
www.chalmers.se

Master's thesis 2020:12

Segmented Classification of Traffic Environments Using RGB-D Data

Considering the effect of image resolution and the relevance of artificial data during training

Johannes Lindgren
Florian Odehnal

Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Adaptive Systems Research Group
Chalmers University of Technology
Gothenburg, Sweden 2020

Faculty of *Konstruktions-, Produktions-, und Fahrzeugtechnik,*
Institute for Systems Theory and Automatic Control (IST)
University of Stuttgart
Stuttgart, Germany 2020

Segmented Classification of Traffic Environments Using RGB-D Data
Considering the effect of image resolution and the relevance of artificial data during training

Johannes Lindgren
Florian Odehnal

Supervisor: Malin Dahl, CPAC Systems AB
Examiner: Peter Forsberg, Department of Mechanics and Maritime Sciences

Master's Thesis 2020:12
Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Adaptive Systems Research Group
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: RGB image overlayed with the corresponding segmentation.

Typeset in LaTeX
Gothenburg, Sweden 2020

Segmented Classification of Traffic Environments Using RGB-D Data
Considering the effect of image resolution and the relevance of artificial data during training

Johannes Lindgren
Florian Odehnal
Department of Mechanics and Maritime Sciences
Chalmers University of Technology

Abstract

In this thesis a method based on previous approaches to perform semantic segmentation using color (RGB) and depth images together (RGB-D) in a Convolutional Neural Network (CNN) is presented. To improve the accuracy of the prediction a fusion module is proposed, to fuse RGB and depth features more efficiently.

Furthermore, it is proved that higher resolution images improve the accuracy of the segmentation, especially for thin structures that are far away. The drawback of increasing the image resolution, on the other hand, is that the runtime increases.

The method is tested using both simulated and real-world data. It is concluded that training the network on artificial data only and then evaluating it using real-world data does not yield a good result due to differences in composition between data. Thus using only artificial data during training is not sufficient. Even though the artificial data can be used for pre-training the network, it is concluded that it does not increase the accuracy compared to training the network using only real-world data.

It is shown that the use of depth images improves the robustness of the segmentation with a large margin. Finally, it is concluded that for this approach to yield its full potential, high-accuracy depth images are a requirement.

Keywords: Semantic Segmentation, Convolutional Neural Networks, Deep Neural Networks, Deep Machine Learning, Computer Vision, RGB-D Data.

Segmentierte Klassifizierung von Objekten mit Hilfe von RGB-D Daten
Unter Berücksichtigung der Auswirkung von Bildauflösung sowie der Verwendung von Simulationsdaten während des Trainings
Johannes Lindgren
Florian Odehnal
Department of Mechanics and Maritime Sciences
Chalmers University of Technology

Zusammenfassung

In dieser Arbeit wird, basierend auf bestehenden Methoden, ein Convolutional Neural Network (CNN) entworfen, um mit Hilfe von Farbbildern (RGB), sowie Tiefenbildern zusammen (RGB-D) Objekte in Bildern pixelweise zu klassifizieren. Um diese sogenannte semantische Segmentierung zu verbessern, wird ein Fusions-Modul vorgestellt, welches die Informationen aus RGB- und Tiefenbild effektiv miteinander verbindet.

Des Weiteren wird die Relevanz von hoch aufgelösten Bildern vor allem zur Wahrnehmung von feinen Strukturen gezeigt. Jedoch mit dem Nachteil, dass sich die Berechnungszeit pro Bild erhöht.

Die entworfene Methode wird sowohl an simulierten als auch an realen Daten optimiert und getestet. Es wird gezeigt, dass Simulationsdaten sich nicht dazu eignen, das CNN zu optimieren und dann an Realdaten zu verwenden. Obwohl ein sogenanntes Transfer-Learning möglich ist, wird aufgezeigt, dass die Optimierung allein basierend auf Realdaten eine größere Genauigkeit bei der Segmentierung erzeugt.

Zudem wird abschließend untersucht, wie die zusätzliche Verwendung von Tiefenbildern die Genauigkeit und Robustheit der Segmentierung beeinflusst. Dabei wird aufgezeigt, dass die Verwendung von Tiefenbildern die Segmentierung verbessert und bei simulierten Störungen deutlich robuster macht, sowie dass Tiefenbilder von hoher Genauigkeit vorteilhaft sind.

Stichwörter: Semantische Segmentatierung, Convolutional Neural Network, Deep Neural Network, Deep Machine Learning, Computer Vision, RGB-D Daten.

Acknowledgements

We would like to thank *CPAC Systems AB* for providing the facilities and equipment to perform this work. In particular we would like to thank Peter Forsberg our examiner and Malin Dahl our industrial supervisor for helping and guiding us through our work despite the tough times that the corona virus has brought. Furthermore, we would like to thank Patricia Pauli, our supervisor from the IST, University of Stuttgart for her guidance and help. We would also like to thank Olav Ravndal Skjølingstad, Torgeir Langfjord Nordgård, Jonas Hellgren and Daniel Weidmann for giving valuable feedback on our thesis writing.

Johannes Lindgren and Florian Odehnal, Gothenburg, June 2020.

Thesis advisor: Malin Dahl
Thesis examiner: Peter Forsberg

Explanation of the authors

We Johannes Lindgren and Florian Odehnal, authors of the master thesis

Segmented Classification of Traffic Environments Using RGB-D Data

ensure:

1. We wrote this thesis on our own.
2. No other sources than the marked sources had been used. All knowledge statements from other works, either literally or analogous, are marked as such.

Date	Johannes Lindgren
Date	Florian Odehnal

Contents

List of Figures

List of Tables

Acronyms

DC-Net *Depth-Context Network.*
DCF-Net 1 *Depth-Context-Fusion Network 1.*
DCF-Net 2 *Depth-Context-Fusion Network 2.*
RGBC-Net *RedGreenBlue-Context Network.*

ASPP Atrous Spatial Pyramid Pooling.

CEL Cross Entropy Loss.
CNN Convolutional Neural Network.
CPU Central Processing Unit.

DNN Deep Neural Network.

eASPP efficient Atrous Spatial Pyramid Pooling.

FLOPS Floating-Point Operations per Second.
fps Frames per Second.

GPU Graphics Processing Unit.

HHA Horizontal disparity, Height above ground and Angle to the gravity axis.

IMU Inertial Measurement Unit.
IoU Intersection over Union.

LiDAR Light Detection And Ranging.

mIoU mean Intersection over Union.

N/A Not Applicable.

pAcc Pixel Accuracy.

ReLu Rectified Linear Unit.
RGB Red Green Blue.
RGB-D Red Green Blue - Depth.

SGD Stochastic Gradient Descent.

WASP Waterfall Atrous Spatial Pooling.

Glossary

ResNet Residual Network (*ResNet-xx*), where *xx* describes the number of total layers in the encoder. The structure of the Residual Network can be seen in the appendix.

artifical data Is data generated from a simulation environment. Synonymous to simulation data.

backpropagation Algorithm to update the weights in a neural network using error gradients.

feature map Output of a convolutional layer, containing information of different features and their position.

training Training of a neural network describes the optimization of the network parameters based on training data as reference.

transfer-learning Technique in machine learning that uses knowledge gained solving one problem and applying it to a different but related problem.

1

Introduction

One influential event in the development of autonomous vehicles was the DARPA grand challenge in 2004, where the task was to drive along a road in the desert [1]. Today, more than 15 years later, technology has come a long way. The challenges have become more complex and require a better understanding of the environment. Especially when it comes to more complicated situations that incorporate other cars and people on the road.

Many companies are running their research on autonomous vehicles. Two such companies are Volvo Trucks and CPAC Systems. One project they are running together is using autonomous trucks for transport in an open-pit mine [2]. Other similar projects are:

- Tesla's autopilot, which is able to drive the vehicle autonomously if supervised by the driver [3].
- Waymo, previously known as Google's self-driving car project, developing self-driving vehicles for daily personal transport as well as self-driving trucks for goods traffic [4].
- Scania's self-driving bus project, where the buses will carry commuters in central Stockholm [5].

Autonomous vehicles have a high potential to increase safety, since they can have 360-degree visual perception and also see parts of the electromagnetic spectrum that humans are unable to see. Since their driving pattern can be optimized in such a way as to minimize energy usage, autonomous vehicles also have the potential to be more environmentally friendly. Furthermore, the automation of commercial vehicles such as trucks offer even more benefits. They can operate in more dangerous environments than vehicles driven by humans since there is no risk of loss of life. Autonomous vehicles can be used 24 hours a day which increases operation time, hence there is also great potential to reduce costs by using them. Because there is no need for a driver the salary cost is also eliminated. Moreover, the trucks can be optimized to be driven in a way that reduces wear on the trucks, which also enables longer maintenance intervals and consequently lowers the maintenance costs.

Important requirements for autonomous vehicles are accurate environment percep-

tion, object detection, and scene understanding since the vehicles would otherwise risk injuring people or damage property. Many different sensors can be used to perceive the environment, such as Light Detection And Ranging (LiDAR) and cameras. LiDARs use lasers to measure the distance to objects in the vehicles' proximity. As an active sensor one major strength of the LiDAR, in contrast to the camera which is a passive sensor, is that it is not light sensitive. Therefore, it can be used at night without any additional light source. However, one drawback of LiDARs is that the point clouds collected by them are usually sparse. The currently commercial available LiDARs are only able to sense a few degrees in a vertical direction and not a full 360-degree spherical view. Another sensor for environment perception is the camera. One of the strengths of using a camera is that it can provide a lot of information because it captures colors, and not only spatial information. Environment information can be extracted from the colors and illumination of the pixels in the image. Moreover, cameras are very cheap compared to LiDARs which makes them an appealing sensor to use for environment perception.

Part of the perception problem using cameras is the data processing needed to extract information about what type of objects exist and where they are in the image. In general, there are two different approaches to extract this information:

- Object detection using bounding boxes around the objects, that are supposed to be detected.

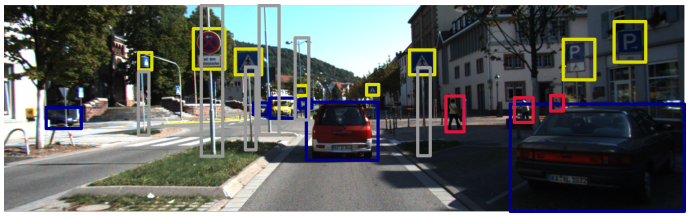


Figure 1.1: Object detection using bounding boxes.

- Semantic segmentation, labeling each pixel of the image with its corresponding class.

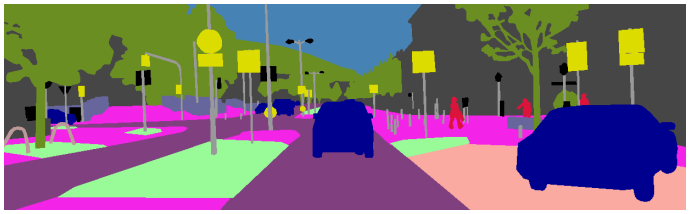


Figure 1.2: Semantic segmentation image.

While object detection using bounding boxes returns position and box size in the image, semantic segmentation returns a high-resolution image specifying every pixel belonging to an object.

To get a rich representation of the segmentation it is beneficial to capture more complementary information. Depth images provide useful complementary information

to the general RGB images. An example of this is a person walking in the shadow of a building. When using an RGB image it is hard to tell the difference between the person and the building due to the poor light conditions. However, using the depth image, it is clear that there is a person in front of the building since it has a different depth than the building. A common way to generate depth images is to use stereo cameras, that is described in Section 2.1.5.

One method to perform image segmentation without depth images is a residual neural network (*ResNet*), that is described in [6], where a Convolutional Neural Network (CNN) is used. The *ResNet* structure is the base of a lot of works on image segmentation such as [7]–[9]. In order to use depth data along with RGB data there are multiple ways to design network structures. One aspect to consider in network design is where and how to fuse the information from RGB and depth data. Multiple approaches such as early, continuous and late fusion of the data have been compared in [10]. Another method to incorporate depth data is depth convolution [11], [12].

1.1 Purpose

The purpose of this project is to investigate how convolutional neural networks can be used to perform semantic segmentation in traffic scenarios in real-time using relatively simple hardware. This is an important aspect since in order to have a major impact on the automotive industry it requires not too expensive hardware. The input to the network should be high-resolution RGB and depth data, which is a recently introduced research area for semantic segmentation. Both the accuracy of the segmentation and the real-time performance will be evaluated, where accuracy is the primary concern. Furthermore, the effect of image resolution will be evaluated in terms of the accuracy of the segmentation. The relevance of artifical data when training networks in contrast to training with real-world data will also be evaluated. Lastly, it is evaluated if the depth data makes the segmentation more robust against disturbances.

1.1.1 Goals

- Design a CNN for semantic segmentation using RGB-D data.
- Achieve a performance of at least 10 Frames per Second (fps) on a Nvidia GeForce RTX 2080 Ti.
- Evaluate the effect of resolution on accuracy.
- Evaluate the relevance of artifical data.
- Evaluate whether depth data makes the segmentation more robust.

The 10 fps specification is chosen since it is a good rule in practise to be able to predict objects at minimum 10 times per second. This is since an autonomous

vehicle will likely update its position based on relative sensors such as an Inertial Measurement Unit (IMU) with a frequency of around 1 kHz. Hence it is suitable to update this relative position estimation with an absolute one with a frequency of 10 Hz. Otherwise the relative position estimation would deteriorate over time.

1.2 Scope

The goal of this thesis is to design a neural network that performs real-time semantic segmentation using RGB-D data. Since there are a lot of approaches using neural networks for semantic segmentation, the task is not to come up with an all-new design or structure but rather base this project on previous works. These works are combined to a network that is used for the segmentation task. The network is trained and evaluated using a desktop computer with a Graphics Processing Unit (GPU) that is provided. This report is also concerned with the preprocessing the data need before it can be used as input to the network. The data used for training are both, artificial and real-world data. However, no new real-world data is created in this project to train on, only existing datasets and artifical data are used.

1.3 Thesis Contribution

The particular contribution of this thesis is a fusion module to combine information of RGB and depth data in an efficient way. There are several approaches for fusion modules [13], [14]. Nevertheless the novel fusion module is supposed to be more efficient, i.e. having less parameters. Furthermore, a study about the effect of the image resolution is contributed. An evaluation of whether artifical data can substitute real-world data when training networks is presented. In a representative robustness study it is shown that depth data is not only complementary to RGB data, i.e. it is increasing the segmentation accuracy, but also redundant to RGB data, which is useful when it comes to disturbances in one of the images.

1.4 Thesis Outline

This thesis consists of six chapters. The current chapter is the first, it is an introduction that gives a brief background to the topic and presents the goal and scope of the work. The second chapter deals with the theory concerning semantic segmentation using CNNs in general, the theory of training CNNs and also with the specific networks used in this work. The third chapter describes the methods used to perform this work and the different datasets are described. The network structure used and the details for the used training setup are also presented. The fourth chapter presents the results of different network approaches. In the fifth chapter the results are discussed. The sixth chapter gives a brief conclusion and suggestions for future works.

2

Theory

This chapter deals with the theory and related works used to perform semantic segmentation. First, the theory used for RGB, then the theory for RGB-D semantic segmentation is described. Finally, the theory to train neural networks is presented.

Semantic segmentation of images is labeling each pixel with a corresponding class. Every pixel is thus given a label such as *car*, *road* or *vegetation* for example as seen in Figure 2.1.

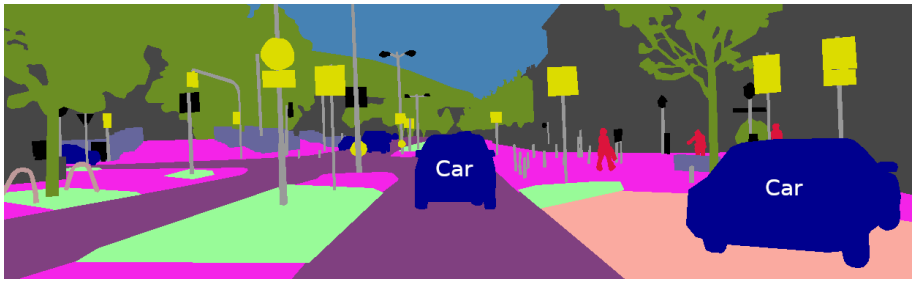


Figure 2.1: Example of a semantic segmentation image.

A further form of segmentation is instance segmentation. For semantic segmentation only the corresponding class is labeled, e.g. a group of people is labeled just as *person*. Instance segmentation, in contrast, distinguishes for some classes between individuals, e.g. instead of giving all persons in the image the same label they use different labels such as person 1, person 2 or car 1 and car 2 as seen in Figure 2.2.

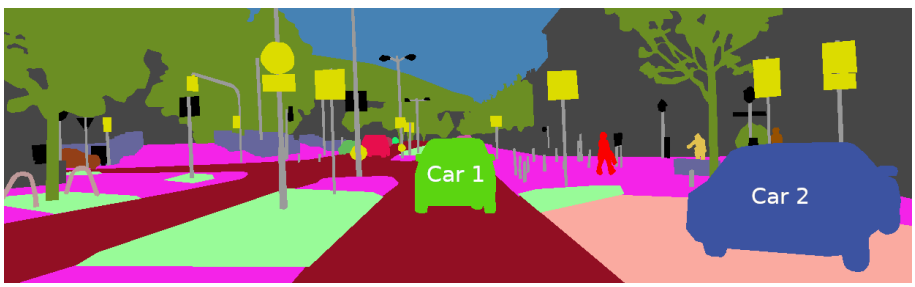


Figure 2.2: Example of an instance segmentation image.

2.1 Previous Work

In this section, previous works on semantic segmentation methods are described. From the methods first used for semantic segmentation up to the modern state-of-the art methods using CNNs as a type of Deep Neural Networks (DNNs).

2.1.1 Hand-Crafted Methods

In the early days, semantic segmentation was done using handcrafted algorithms. The image was processed with different filters that are able to detect edges or other structures. These simple structures are combined to more complex structures. Two of the most well-known handcrafted filters are the Sobel and Roberts operator. They are small matrices with positive and negative values ordered such that edges are detected in an image if the filter is convoluted with the image [15]. An example of the Sobel filters for horizontal and vertical edges can be seen in Figure 2.3.

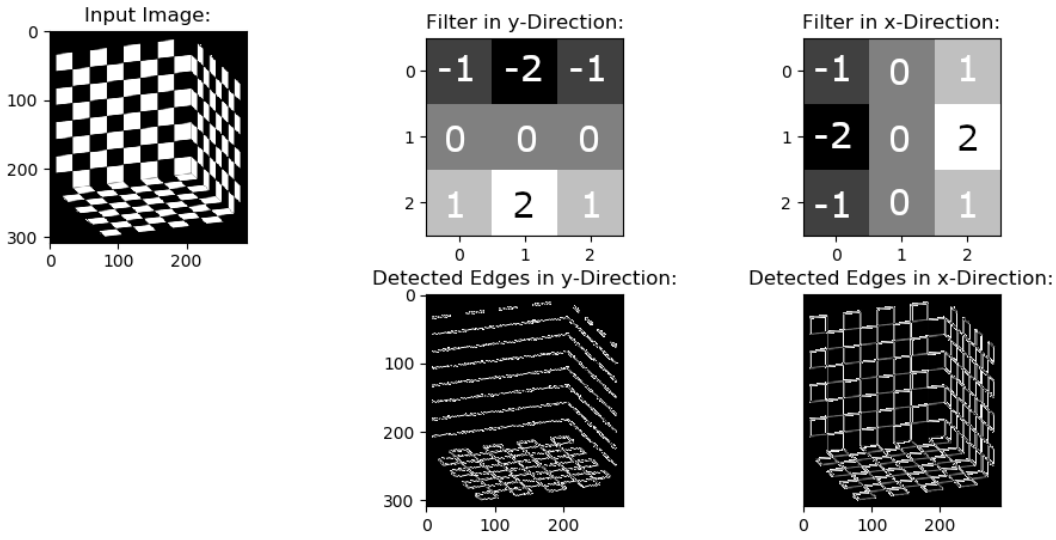


Figure 2.3: Example of edge detection using Sobel filters.

Another property of images exploited by handcrafted methods is the relation between pixels. Nearby pixels in an image are more likely to have the same label, unlike pixels that are far away from each other. Pixels with similar colors are also more likely to have the same label compared to pixels with different colors. Moreover the relation between different objects in the image is used. For example, it is more likely that a person is on the sidewalk than being up in the sky [16].

A major disadvantage with handcrafted methods is that they are very scene dependent and need to be adjusted or changed for each use case by hand. CNNs, further explained in Section 2.1.3, are also dependent on the data they are trained on, but since CNNs can be trained on large datasets they are less scene dependent. Furthermore due to the fact that CNNs are deeper, i.e. contain more filter layers than handcrafted methods, they have the ability to generalize beyond specific datasets.

2.1.2 Deep Neural Networks

DNNs are essentially neural networks with many layers. Neural networks get their name from the fact that they try to imitate the function of neurons in the brain. They consist of neurons connected to each other, often divided into layers. One of the simplest network structures is a fully connected feed-forward structure, as shown in Figure 2.4. The structure shown in the example consists of four layers. One input layer, two hidden layers, and one output layer. The output from each neuron is calculated according to Equation (2.1), where α_i are the weights that are multiplied with the output y_{i-1} from the previous layer $i-1$ and β_i is the bias term that is added before used as input z to the activation function f. One common activation function is the Rectified Linear Unit (ReLu)-function in Equation (2.2).

$$y_i = f(z) = f(\alpha_i y_{i-1} + \beta_i) \tag{2.1}$$

$$\text{ReLu} = \max(0, z) \tag{2.2}$$

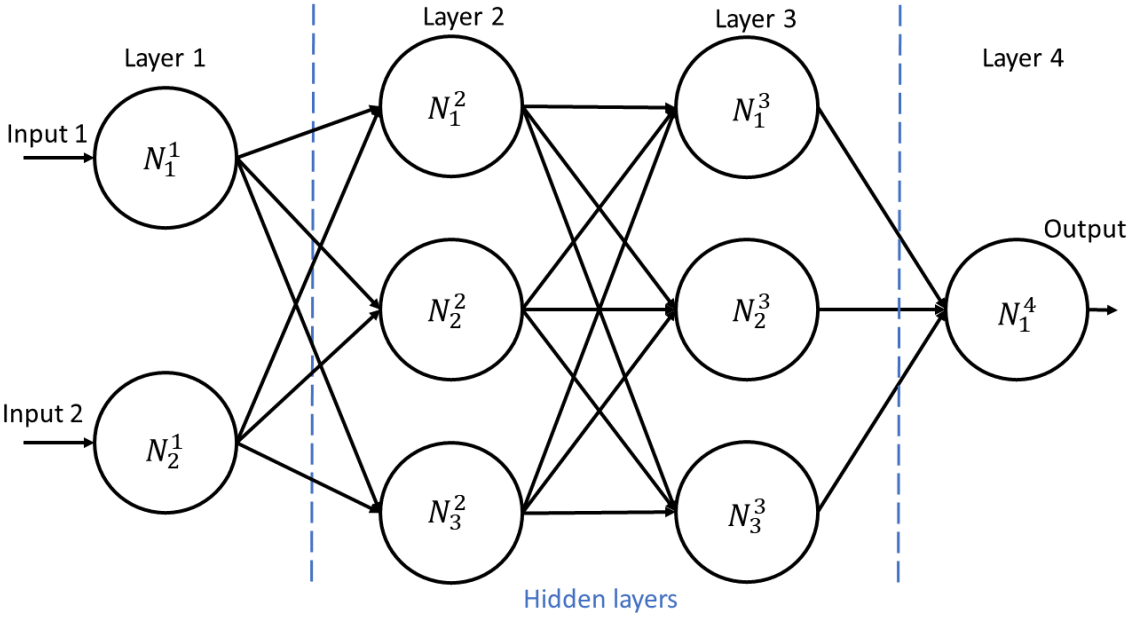


Figure 2.4: Neural network structure with 4 layers.

The weight and bias values of the activation functions need to be optimized such that the DNN is processing data to a certain output. To do this optimization or training of the network there are two main strategies. The first strategy is *supervised learning*. It takes datasets consisting of pairs with input data x and the corresponding ground truth output y, that should be generated based on the input. To train the network the error between y and the prediction of the DNN $\hat{y}$ is minimized with backpropagation using gradient descent algorithms. In the specific use case for DNNs the error function is called loss function.

The second training strategy does not use labeled data and is called *unsupervised*

learning. *Unsupervised learning* is usually used if there is an associated cost function to the performance on a task. In the case of image segmentation there is no such cost function so *unsupervised learning* is rarely used for semantic segmentation.

2.1.3 Convolutional Neural Networks

A CNN is a type of DNN that contains convolutional layers. A convolutional layer takes a matrix of values as input and shifts a filter, which is a smaller matrix, over the input image matrix. An example of a convolutional layer can be seen in Figure 2.5.

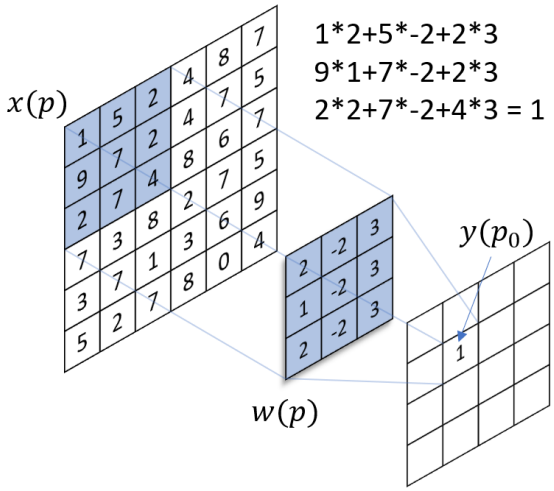


Figure 2.5: Convolutional layer with zero bias.

The convolutional layer as matrix multiplication is described by Equation (2.3).

$$y(p_0) = b + \sum_{p_n \in R} w(p_n) x(p_0 + p_n), \tag{2.3}$$

where p_0 is the current pixel, p_n is the relative pixel, $R \in \mathbb{Z}^2$ is the filter, $w(p_n) \in \mathbb{R}$ is the weight of the filter on its pixel p_n, $x(p_0 + p_n) \in \mathbb{R}$ is the pixel value of the input and $b \in \mathbb{R}$ is the bias. This matrix operation is done for each pixel in the image.

The difference between hand-crafted method filters and CNN filters is that the CNN filters are trained over gradients of the loss function with ground truth data to get the filter weights instead of defining them manually. So in general hand-designed algorithms follow the same structure as trained convolutional layers of a CNN. But since the CNN is optimizing its filters on a large amount of data it is able to extract more specific features. Besides, it can be designed and optimized with more filter layers than hand-crafted algorithms usually contain. The training procedure for CNNs is described later in Section 2.5.

Another type of layer often implemented in a CNN is a pooling layer. It reduces the size of the input matrix by shifting a matrix over the input based on different pooling approaches. The pooling layer either calculates the mean value of a matrix area or the maximum value and takes this value for the output of the pooling operation. An example of a maxpooling can be seen in Figure 2.6. It can be seen as a downsampling layer of the network. The opposite is an unpooling layer, upsampling the size of a feature map again [17].

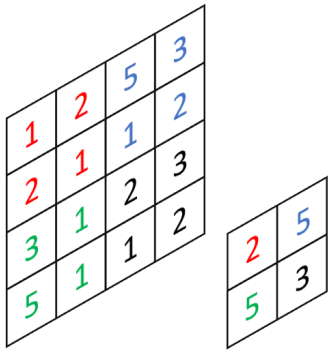


Figure 2.6: Maxpooling layer, original matrix and the resulting matrix. Filter size (2×2) and stride 2.

CNNs can be used to classify whole images to one class, classify objects and their position in an image, or to classify every pixel in an image to a certain class which is the intention of this thesis. The structure of the used CNN depends on the task.

2.1.4 Instance Segmentation

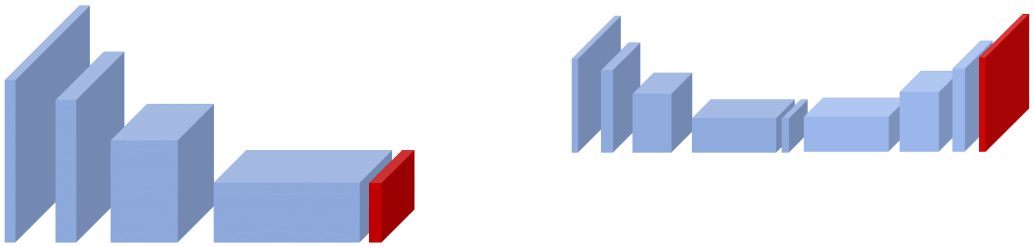
Current state-of-the-art networks for instance segmentation are R-CNN (Recurrent-Convolutional Neural Network) [18]. They first detect boxes around objects and then perform semantic segmentation on these boxes to detect the instances of the class. Performing both object detection followed by semantic segmentation is computationally expensive. Thus it has not been implemented or tested for real-time segmentation, which is the attempt of this work. Hence this approach is not used.

2.1.5 CNN for RGB Image Segmentation

Semantic segmentation of RGB images can be done in different ways using CNNs. Although all these networks use the same base modules and methods, the difference between the networks is the architecture that is built out of these base modules. Thus in this section the most important features of existing design methods are presented and compared.

Semantic segmentation is supposed to extract high-resolution outputs. For pure classification of an image the network is decreasing the size of the image down to rich features using convolution and pooling operations. These features are combined to an output of the size of the number of classes representing the probability of

the image belonging to each class. On the other hand, for semantic segmentation, the output size of the prediction has to be as large as the input size. However semantic segmentation networks use convolutional layers and pooling methods to detect features in the image. In this way the resolution is decreased, but it needs to be increased again to the original size for the semantically segmented output. A structure used for CNNs to get back to the original input size is an encoder-decoder structure that can be seen in Figure 2.7 (b). In an encoder-decoder structure, the encoder extracts features from the images and combines them to rich features. If an encoder is combined with a classification layer it is able to perform pure classification of images, this type of structure is seen in Figure 2.7 (a). To get back to the original size there is a decoder with upsampling, and deconvolution layers, performing inverse convolutions.



(a) Encoder structure, with a red classification layer for image classification.

(b) Encoder-decoder structure for semantic segmentation, with a red output layer.

Figure 2.7: CNN structures for image classification and image semantic segmentation.

A problem using an encoder-decoder structure is the loss of spatial information due to the compression of feature maps to a smaller size, e.g. 32 times smaller, when using a *ResNet* encoder. Thus when performing semantic segmentation there are methods to keep the spatial information.

U-Net [19] introduced shortcuts between the encoder and decoder to keep spatial information. The structure is seen in Figure 2.8. With these shortcuts high resolution spatial information of the early convolution layers in the encoder are added to the output of the deconvolution layers in the decoder. Thus the information about the position of low level features is kept and the spatial information is not only dependent on the low resolution output at the end of the encoder.

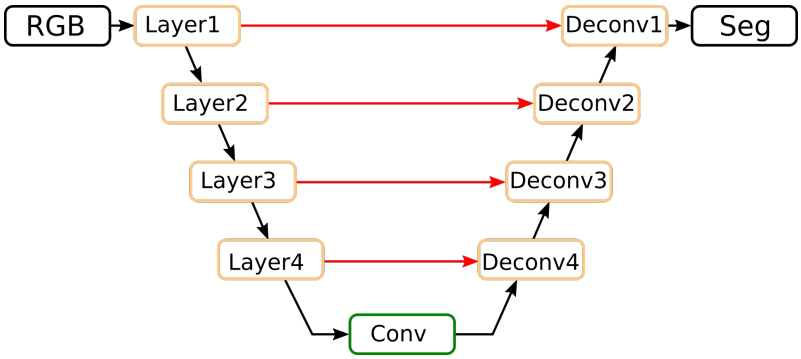


Figure 2.8: Structure of the U-Net with shortcuts marked by red arrows, to keep spatial information of objects [19].

The resolution of objects at different distances from the camera is another issue. While objects close to the camera have a high resolution, the same object further away has a lower resolution. To adapt a network, such that it is able to classify both objects with high and low resolution to the same class, dilated convolutions are used. Dilated convolutions are explained in the following paragraph.

Deeplab [20] is introducing atrous convolution and atrous spatial pyramid pooling layers (ASPP-layers). An atrous convolution is a convolution with dilation without increasing the number of parameters. This dilation is made by leaving out pixels in between. Thus for example a (3×3) filter with 9 parameters is enlarged to a (5×5) filter still with 9 parameters if a dilation rate of $r = 1$ is applied as shown in Figure 2.9. With dilation the receptive field is enlarged and thus with higher dilation rate r, the detected feature of a feature layer is allowed to have a larger size. For lower resolution scenes, or objects a smaller rate is more effective. Thus to extract high resolution, low resolution and medium resolution sized objects different dilation size filters are needed. The ASPP-layer takes an input feature map with different scene resolutions and performs atrous convolutions with different dilation rates. These outputs which contain different resolution information are concatenated to one output of the ASPP layer, this is seen in Figure 2.10.

Equation for dilated convolutions, with the same notation as in Equation (2.3) and $r \in \mathbb{Z}^+$ representing the dilation rate:

$$y(p_0) = \sum_{p_n \in R} w(p_n) x(p_0 + r \cdot p_n) \tag{2.4}$$

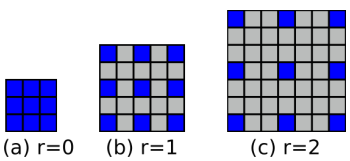


Figure 2.9: (3×3) convolution filters with different dilation rates r.

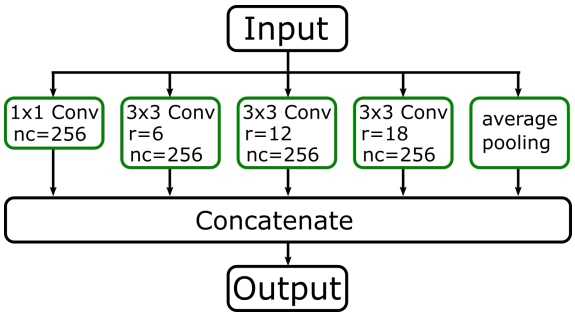


Figure 2.10: ASPP structure [20]. $n \times n$ Conv is a Convolution with filter size n, r is the dilation rate and nc is the number of channels.

AdaptNet++ [14] improved the ASPP layer to make it more efficient. Here the dilated convolutions with different rates are replaced with a (1×1) convolution down to less channels than the original ASPP, followed by two dilated (3×3) convolutions and after this a (1×1) convolution back up to the origin channel number. The adapted ASPP structure contains 87.87% less parameters and still achieves a higher accuracy than the original proposed ASPP-layer [14]. By using two dilated convolutions in a row they increase the receptive field additionally compared to a single dilated convolution. The default ASPP and the efficient Atrous Spatial Pyramid Pooling (eASPP) layer can be seen in Figure 2.10 and Figure 2.11 respectively.

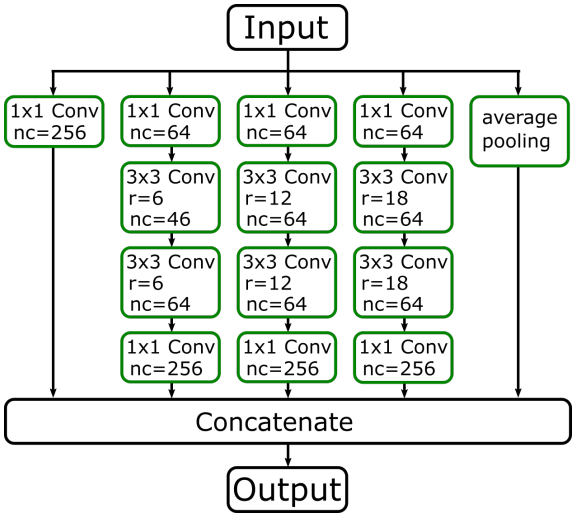


Figure 2.11: eASPP structure [14].

WASP-structure: Waterfall Atrous Spatial Pooling is another way to adapt the ASPP layer. It is described in [21] and can be seen in Figure 2.12. While the ASPP performs only parallel convolutions the WASP-layer also uses the cascade structure, that can be seen in Figure 2.13 (b). By using the cascade structure, the WASP layer combines the larger field of view from traditional ASPP with the cascade approach used in [22], that enables the network to capture long range information.

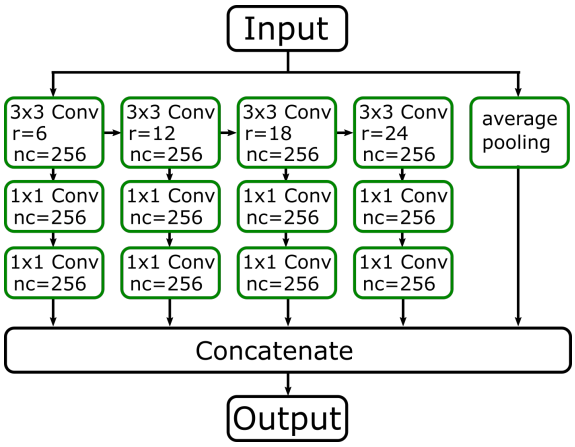
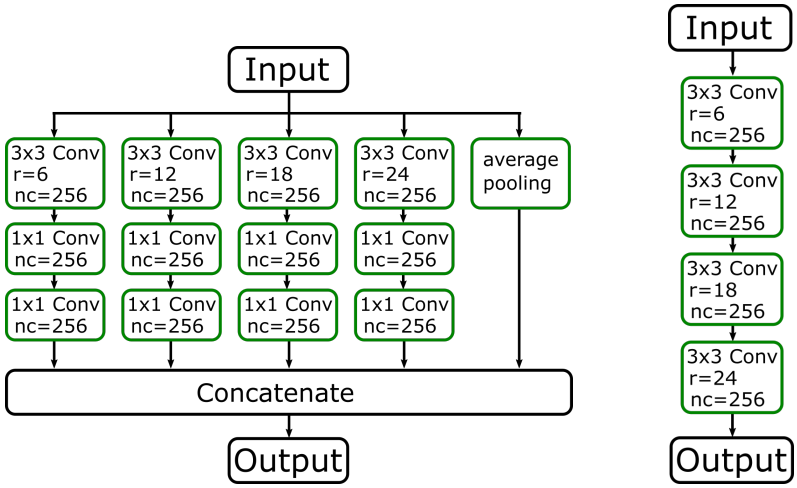


Figure 2.12: WASP structure [21]

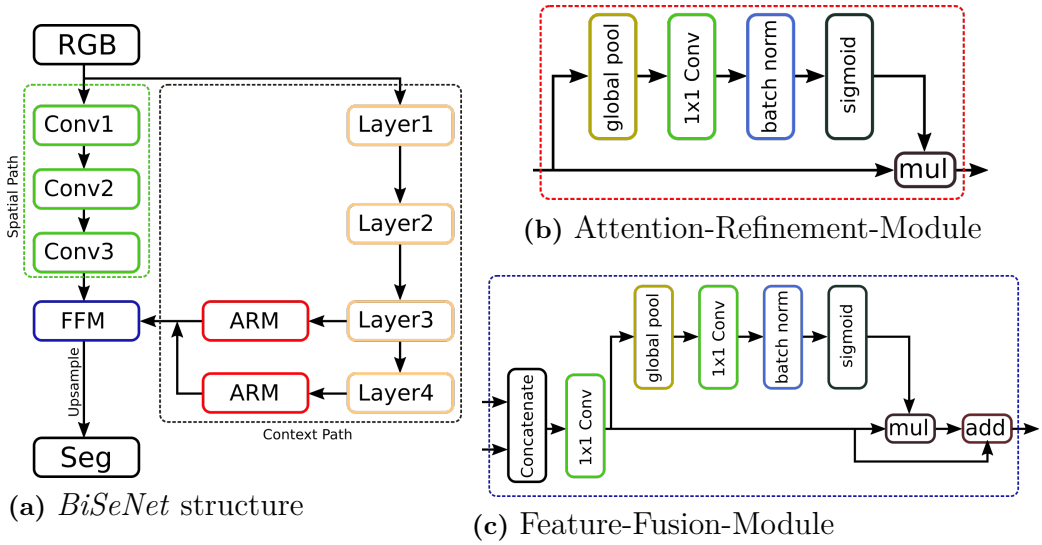


(a) Adapted ASPP structure.

(b) Cascade structure.

Figure 2.13: ASPP structure compared to the cascade structure [21].

BiSeNet, the Bilateral Segmentation Network [7] is a lightweight network structure designed for fast real-time segmentation, the general architecture can be seen in Figure 2.14 (a). It tries to overcome the problem of different resolutions and loss of spatial information by using two branches. One branch is to extract spatial information out of the image. In this path the feature maps are not pooled down to enlarge the receptive field as it is usually the case in encoders. Thus the resolution of the feature maps is kept high. The second branch is to extract context information out of the image. Here a *ResNet-50* [23] is used as encoder to get a larger receptive field and thus gather more context information. At the end attention-refinement modules are used to weigh the features of each stage, see Figure 2.14 (b). This attention-refinement module performs global average pooling to capture the importance of each channel and uses attention by multiplying each channel with this value to guide the feature learning. To fuse these two branches in the end of the *BiSeNet*



(a) *BiSeNet* structure

(b) Attention-Refinement-Module

(c) Feature-Fusion-Module

Figure 2.14: *BiSeNet* structure and modules.

a feature-fusion module is used to fuse the information of both branches in a proper way. This structure can be seen in Figure 2.14 (c). One advantage of this network structure is that it is simple and thus does not need as much computations as most of the other networks. On the other hand the *BiSeNet* uses no decoder and is thus harder to optimize.

2.2 Performance of Networks

To improve the speed of a network for real-time semantic segmentation there are mainly two approaches:

- Decrease the image size.
- Pruning of nodes with small weights.

2.2.1 Decrease Image Size

The most simple method is to crop or resize images to a smaller input size, such that the network has to perform less computations for each image. If the image is cropped, information such as context might be lost, thus it is not considered as useful method to speed up the network. Resizing the image means that high-resolution information is lost such as small and thin structures that are far away. A short study to prove this assumption is presented in Section 4.2.

2.2.2 Pruning

Another method especially used for fully connected DNNs is to prune neurons whose function weights are close or equal to zero and thus have almost no influence on the

output. The first step is to fine-tune a network until convergence on the target task. Then some neurons are pruned and the network is fine-tuned further. The pruning is stopped after the accuracy decreases, or if a wanted number of parameters is achieved. One difficulty with the pruning is to find the right pruning criterion. A common criterion is to start with the nodes with the smallest weights. As previously mentioned they do not deliver meaningful or important parts to the total sum for the fully connected layers. In CNNs it is different, some weights in the convolutional layers are small but still useful (compare *Sobel filter*) to detect structures. One method to perform pruning on CNNs is described in [24]. The way to prune CNNs is to neglect whole filters and thus channels that have small values for their feature maps.

2.3 Depth Images

Before describing the usage of RGB-D images it is important to understand what the depth image represents and how it is captured. A depth image represents the distance of every pixel to the camera in an image. There are two different ways to measure the distance to the camera depending on how the images are captured:

- Distance to the camera as a point.
- Distance to the camera plane parallel to the image.

Using images from a stereo camera, the depth is calculated out of the disparity between key-points of two stereo images as seen in Figure 2.15 below. Key-points are for example edges and corners of significant objects.

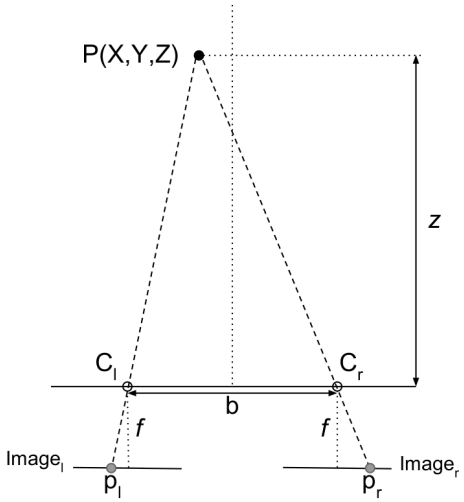


Figure 2.15: Geometry for the calculation of the depth z to the camera plane in a 2D example [25].

With geometric formulas the depth $z \in \mathbb{R}$ is calculated out of the disparity of the point projections p_l and $p_r \in \mathbb{R}^2$ of the key-point $P(X, Y, Z) \in \mathbb{R}^3$ on the images of

the left and the right camera. Both cameras are considered to be optimal pinhole cameras with centers C_l and C_r. The cameras have the same focal length $f \in \mathbb{R}$, which is the distance between the lens of a camera and its sensor, and baseline $b \in \mathbb{R}$, as distance between the centers of the cameras. Thus the depth z can be calculated as shown below:

$$z = \frac{bf}{p_l - p_r} = \frac{bf}{d(p)} \quad [25] \tag{2.5}$$

Where $d(p) \in \mathbb{R}$ is the disparity between both image points p_l and p_r. In this way the values of the depth image represent the distance between the object and the plane parallel to the image planes.

To generate the disparity it is necessary to find key points in the image, and compare their location in the left and right image. Such a generation of depth images out of stereo images was described and done in [25].

Another approach to generate depth images for autonomous vehicles is depth estimation based on mono images taken when the vehicle is moving. Several images while driving are compared in a CNN. If there are key-points moving fast, they are likely to be close to the camera. If they are moving slow they are more likely to be further away from the camera. Thus a high density depth map can be generated. This has been done in Monodepth2 [26].

In the simulation environment Carla [27], which is described in Section 3.1.1, the depth is also represented as distance between the object and the camera plane. But it is encoded in RGB space. Thus the image needs to be preprocessed to a depth image holding only values for depth and not for red, green and blue. The decoding is made in the following way:

$$depth_{normalized} = \frac{R + G \cdot 256 + B \cdot 256 \cdot 256}{(256 \cdot 256 \cdot 256 - 1)}, \tag{2.6}$$

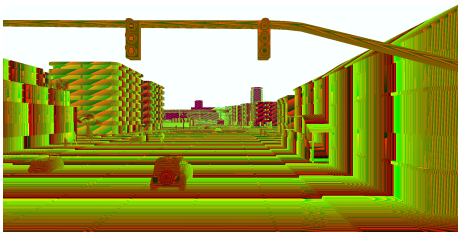
where R, G, B are the red, green and blue color channels respectively. This returns a normalized value space from 0 to 1 where 1 equals the furthest measured distance, which is for Carla set by default on 1000m. All values greater than 1000m are set to the value 1000m. To focus the depth image more on closer objects it is scaled to a logarithmic scale with the formula

$$depth_{log} = 1 + log(depth_{normalized}). \tag{2.7}$$

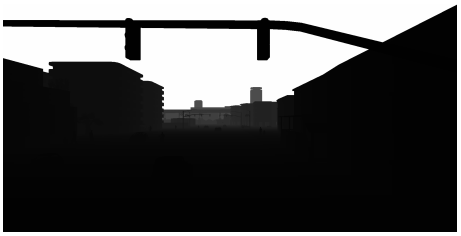
This preprocessing can be seen in Figure 2.16.

Since Carla is a simulation environment, the created depth maps have a high quality. The generated depth images from stereo cameras are not of the same quality. To

improve the quality of depth images there are approaches to use sensor fusion of cameras and LiDARs. But the point cloud LiDARs produce is typically sparse. The downside with calculating depth from images is that poor lighting conditions can render inaccurate depth estimations. Hence by fusing the two data sources the depth image quality is improved. The fusion of LiDARs is done in [28] based on CNNs and feature extraction of key-points.



(a) RGB-encoded Carla depth image.



(b) Normalized depth image.

(c) Depth image in logarithmic scale.

Figure 2.16: Processing steps of a RGB encoded depth image from Carla to a logarithmic depth image.

2.4 CNN for RGB-D Image Segmentation

One of the most important questions using depth images along with standard RGB images is the way the information from both images are fused together. In general the treatment of RGB and depth images in one network can be divided into different architecture structures. These architectures can be seen in Figure 2.17.

Networks structured as shown in Figure 2.17 (a) concatenate the RGB and depth input to one input consisting of four channels. These four channels are then fed through a CNN. Thus this structure is called early fusion structure.

Network structures similar to Figure 2.17 (b) perform semantic segmentation on a RGB and depth branch separately and fuse the output of each CNN in the end, which is called late fusion.

Networks built after the structure seen in Figure 2.17 (c) encode the RGB and depth input separately, transform the rich features of the encoders together and perform

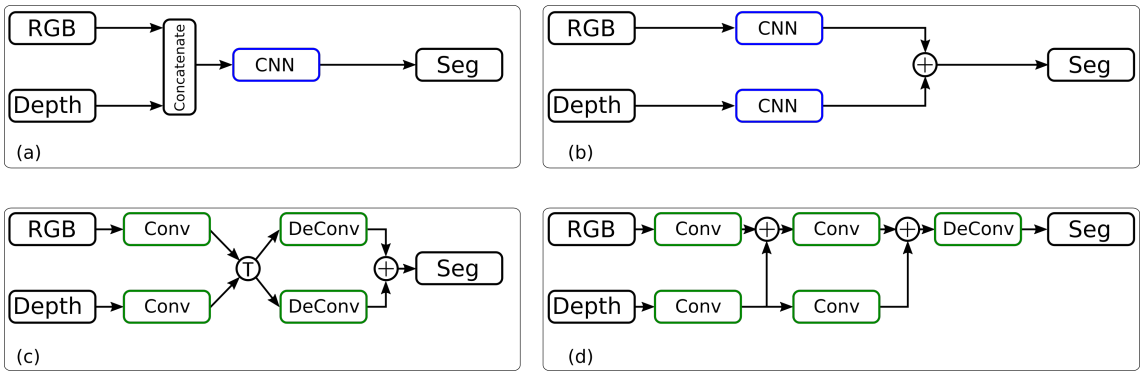


Figure 2.17: Different semantic segmentation network architectures to fuse RGB and depth data.

deconvolutions to the original image size for segmentation in two decoders whose outputs are added to one semantic segmentation output.

Networks like Figure 2.17 (d) fuse the RGB and depth feature layers in steps. Thus the depth features are continuously fed into the RGB branch such that the depth features are also recognized in the RGB encoder.

2.4.1 Early Fusion of RGB and Depth Data

The ***CFN-Net*** in article [29] is one of the first approaches to improve semantic segmentation of RGB images with the use of depth images. The depth images are used to detect higher scene resolutions. Such as areas close to the camera where objects are usually larger in resolution, as well as lower scene resolutions, meaning areas further away with lower object size in the image. The image is then split into layers i.e. contexts with scenes close to camera, middle distance and background layer. These layers are then fed through different branches that use filters with different dilation rates and thus receptive field sizes, larger ones for closer scenes and smaller ones for scenes farther away. A problem with splitting up the input according to depth is to identify objects spanning the full depth from close to the camera to deep in the space, such as roads.

In article [12] a ***depth-aware CNN*** fusing the RGB-D data from beginning (compare with network structure in Figure 2.17 (a)) is described. The network consists of depth-aware convolutions instead of standard convolutions and depth-aware average pooling instead of normal average pooling. This enables the network to make use of the depth data in the entire network.

Depth-aware convolutions are calculated as follows:

$$y(p_0) = b + \sum_{p_n \in R} w(p_n) F_D(p_0, p_0 + p_n) x(p_0 + p_n) \tag{2.8}$$

Where:

$$F_D(p_i, p_j) = \exp\left(\alpha |D(p_i) - D(p_j)|\right) \tag{2.9}$$

with the same notation as in Equation (2.3) and $D(p_i)$ representing the depth value of pixel p_i. The depth awareness can be scaled by the scaling factor $\alpha \in \mathbb{R}$.

These depth-aware convolutions result in lower accuracy than continuous fusion approaches [11], [30].

2.4.2 Continuous Fusion of RGB and Depth Data

The more recent approaches to perform RGB-D image segmentation are to use a continuous fusion of the RGB and depth feature maps. They usually consist of two encoders, one for the RGB and one for the depth images. The encoder feature maps are fused after each bottleneck layer of the encoder. According to the accuracy numbers in the corresponding papers [11], [12], [31], [32] this method is more efficient and keeps more additional information of the depth image than fusing it in the beginning like the depth-aware convolution does.

ACNet [31] is a network where the RGB and depth branches are fused continuously in steps as seen in Figure 2.18. It is using two encoders, one for the RGB and one for the depth data. The encoder is a *ResNet-50* encoder with four layers and a preprocessing convolution. The output after each encoder layer is fused in a middle branch to combine information of the RGB and the depth image. The output of the middle branch is fed through a decoder with upsampling operations to create the original resolution size of the input. Another introduction in paper [31] is the use of an attention module *ACM*, which can be seen in Section 3.2.2. It calculates the importance of each channel and then multiplying each channel with this importance factor. Thus the network has a higher attention on the channels that are more important.

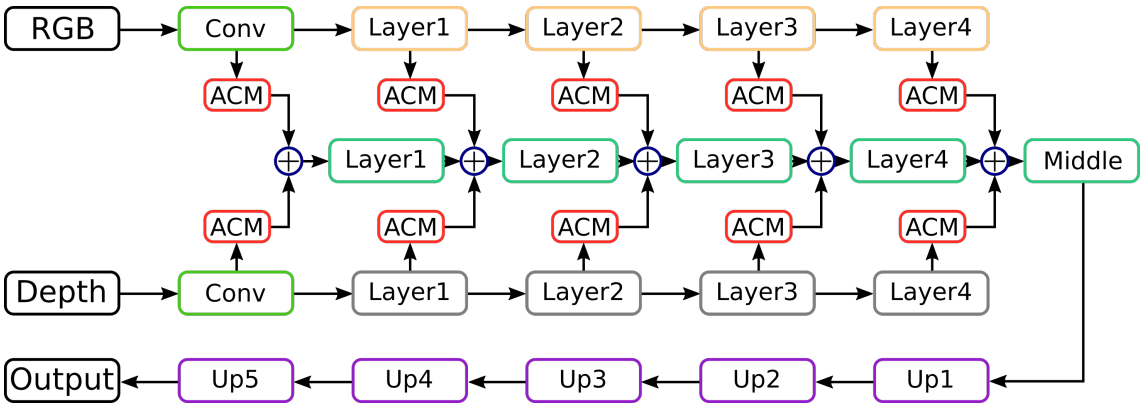


Figure 2.18: Network structure of the *ACNet* [31].

RedNet [32] is using a structure similar to Figure 2.17 (d) and can be seen in Figure 2.19. It uses the *ResNet-50* as encoder and residual blocks with transposed convolutions as decoder. If a network is deep it is possible that for a lot of activation functions the gradient becomes small, thus the gradients are vanishing. Multiplying these small gradients up through backpropagation the effective gradient becomes close to zero and thus the step size during training becomes virtually zero. To

overcome this gradient vanishing problem in the *RedNet* different stage outputs from the decoder are used during training. They are downsampled compared to the end output, which has the size of the ground truth image. Thus the ground truth images are scaled down to the downsampled output size and these are also used in the loss function for training. Another method that prevents gradient vanishing are the residual connections in the network. The *RedNet* uses an upsample residual block in the decoder instead of performing only upsampling and transposed convolutions. These structures are described more in detail in Section 2.5.4.

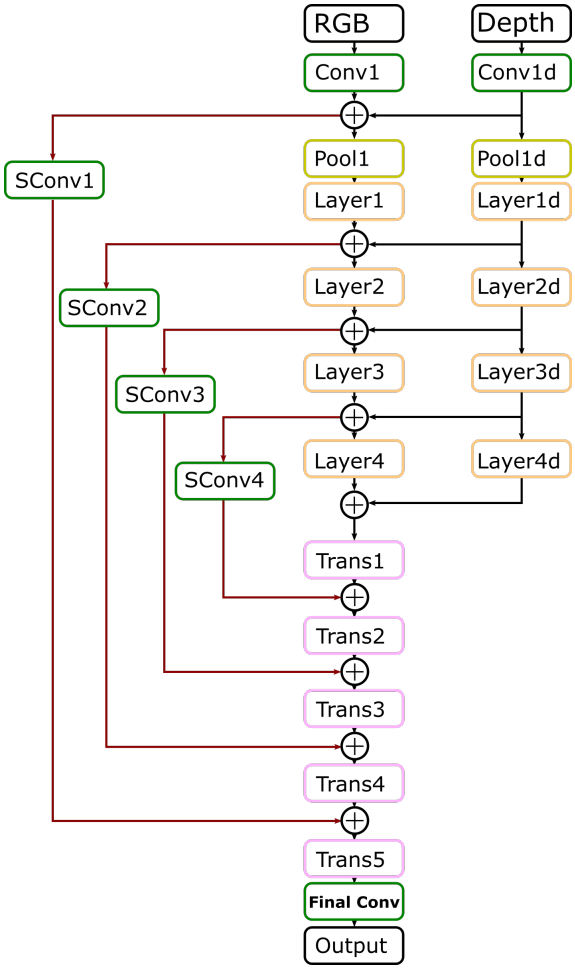


Figure 2.19: Network structure of the RedNet[32].

The network structure used in the ***RDF-Net*** [13] uses a *ResNet-101* as encoder and Refine-Net modules as decoder structure to fuse the different resolutions of the encoder. It contains multi-modal feature fusion modules to fuse the depth and the RGB information. Unlike the *AdaptNet++*, presented in the next paragraph, it does not use a concatenation and a sorting algorithm, but different convolutions and *ReLU* functions before and after summing the feature maps of the RGB and depth branch. The different modules and the complete network structure of the *RDF-Net* can be seen in Figure 2.20 and 2.21. Furthermore there is more than one way to encode the depth image. In [13] a method of encoding depth using three channels Horizontal disparity, Height above ground and Angle to the gravity axis

(HHA) is used. Since the HHA encoding is calculated from the depth image it does not provided any additional information. Hence a deep neural network could learn these features.

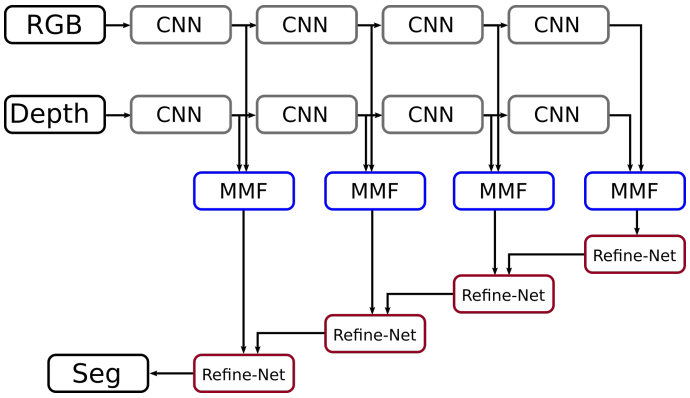


Figure 2.20: Network structure of the *RDF-Net*[13].

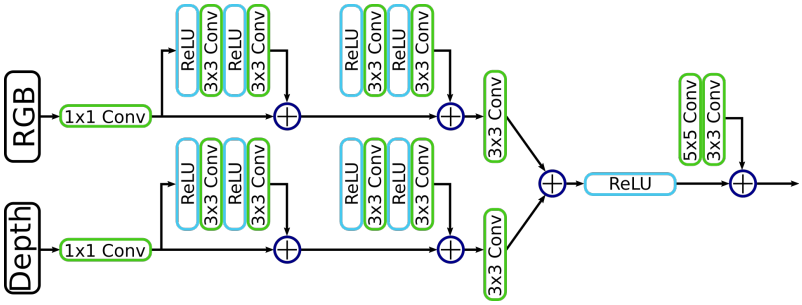


Figure 2.21: Multi-modal feature fusion module (MMF) of the *RDF-Net*[13].

In ***AdaptNet++***[14] a fusion module to fuse information from the RGB and depth branches is proposed. The fusion module first concatenates the channels of both branches. Then weighs each channel according to its importance, the structure is seen in Figure 2.22. This less complex structure improves the speed of the network compared to other fusion structures like the *RDF-Net*'s multi-modal feature fusion of the *RDF-Net*. The *RDF-Net*'s multi-modal feature fusion contain a lot of parameters and thus requires a lot of FLOPS when processing.

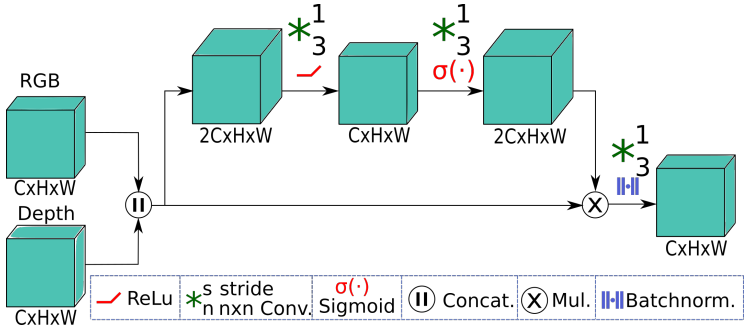


Figure 2.22: AdaptNet++ fusion module for RGB and depth feature maps [14].

2.4.3 Comparison of Fusion Modules

The straight forward way to fuse modalities is to add the corresponding feature map on the same stage. This can be done if the feature maps that are added represent the same features. The *RDF-Net* for example uses residual and convolution layers before and after the summation to improve the fusion of the modalities, but with a high computational effort.

The second way to fuse different modalities is to concatenate the feature maps of the RGB and depth branch. This will prevent the summation of two different feature maps which do not represent the same features. Because of the concatenation the number of channels is not the same as before thus the output of the fusion module is too large for the next convolution layer. Therefore a lot of approaches use attention based structures where each channel of a feature map is multiplied with attention values, followed by convolution layers to get back to the input channel size. The *AdaptNet++* uses attention values for each pixel in each feature map. Other networks like the *BiSeNet* (even though they fuse two RGB feature maps) use attention values for the whole feature maps. This is done by applying global pooling operations on each feature map, followed by (1×1) convolutions to get correlation between the different feature channels and a sigmoid function as activation function. Then each feature map is multiplied by its corresponding attention value.

2.5 Training of CNNs

As mentioned in the beginning of Chapter 2, CNNs need to be trained to yield good results on a certain task by generalizing information extraction. The goal during training is to get as close as possible to the ground truth data with the corresponding input data. Therefore the parameters, $\Theta \in \mathbb{R}^n$, which are the weights of a convolution filter as well as the weights of the activation functions need to be tuned.

It is necessary to measure how much the prediction of the network $\hat{y}$ differs from to the ground truth data y. Therefore a objective function describing the error is the loss function $J(\Theta, \hat{y}, y) \in \mathbb{R}$. Specific loss functions for semantic segmentation CNNs are described in detail in Section 2.5.5.

2.5.1 Optimization Algorithms

There are different optimization strategies to minimize the loss function and thus find the optimal parameters. These strategies can be used to train any DNN. The base approach is the *gradient descent algorithm* as a first-order gradient optimization algorithm for the optimization problem described in Equation (2.10) [33].

$$\Theta^* = \underset{\Theta}{\operatorname{argmin}}\, J(\Theta, \hat{y}, y) \tag{2.10}$$

To find the optimal parameters Θ^*, the *gradient descent algorithm* is applied iteratively. It uses a fixed learning rate $\eta \in \mathbb{R}$ as shown in Equation (2.12). The algorithm ends if a threshold area for Θ is reached.

$$g_t = \nabla_\Theta J(\Theta, \hat{y}, y) \tag{2.11}$$
$$\Theta_{k+1} = \Theta_k - \eta g_t \tag{2.12}$$

Since the gradients $g_t \in \mathbb{R}^n$ of the loss function are dependent on the input given to the network the gradients vary for each input sample. Therefore a training strategy is to calculate the gradient for several inputs of a batch. Then the mean gradient over all these gradients is calculated. This batch calculation is done to increase the probability of finding the global optimum of the loss function. This method is called batch gradient descent. A too large batch size is computationally expensive and can especially for CNNs reach memory boundaries, since the gradient needs to be stored for each input of the batch.

For the Stochastic Gradient Descent (SGD) algorithm the gradient direction of the gradient descent algorithm gets replaced with a random direction with a specified probability. This is helpful in case the algorithm gets stuck in a local minimum that is not the global minimum. Thus unlike the batch gradient descent, the SGD will most likely find the global minimum, if the learning rate is slowly decreased. If the learning rate stays at a fixed value, SGD will end in an area around the optimum.

The SGD algorithm can be extended with momentum as shown in Equation (2.13) and (2.14). The momentum v_k is a moving average gradient over a few time steps weighted with the factor γ. It helps to move faster towards the optimum in case the local gradient is not pointed straight to the optimum but the average gradient over a few steps is. Thus it prevents the algorithm from oscillating too much around the true direction towards the optimum during training.

$$v_k = \gamma v_{k-1} + \eta g_t \tag{2.13}$$
$$\Theta_{k+1} = \Theta_k - v_k \tag{2.14}$$

2.5.2 Normalization of Data

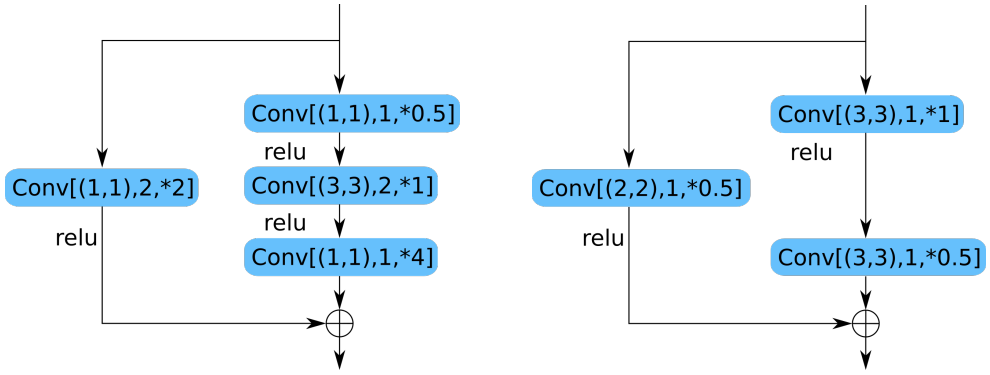
When training CNNs on images it is common to normalize the data to a mean of zero and a standard deviation of 0.225 to improve the training convergence and results [34]. This is also done for a lot of available pre-trained networks such as the *ResNet* encoders. The normalization of images also makes them less light dependent i.e. the same scene under two different lighting conditions will have very different pixel values for unnormalized images. But if the images are normalized the values are similar and thus independent of the original light setting which makes the predictions more robust.

2.5.3 Batch Normalization

Batch normalization is used to speed up training. In a neural network it is possible that the output of some activation functions are not equally distributed and scaled for the parameters. Thus some parameters are not trained as good as others, since the learning rate is too large for some parameters and too small for others. During batch normalization the output of the previous activation function is normalized, meaning it shifts the output of each batch, that is processed during training, by mean and scales it by standard deviation. This enables the use of higher learning rates without overshooting the optimum and acts as regularization as well, since it adds a small noise to the activation function. The regularization in a neural network also prevents overfitting on training data.

2.5.4 Bottleneck and Inverted Bottlenecks in Residual Structures

Bottlenecks in CNNs are used to decrease the size of the feature maps and increase the number of channels. To increase the efficiency of training there are bottlenecks with residual structures to overcome the gradient vanishing problem in DNNs [23]. In a residual structure the input is kept and added to the convolution output of the layer thus the output of the layer is $y = F(x) + x$. Thus on backpropagation the gradient is always $\frac{dy}{dx} = \frac{dF(x)}{dx} + 1$, with 1 from the residual part. The *RedNet* described in Section 2.4 contains residual bottlenecks, non-bottlenecks and inverted bottlenecks. A visualization of residual structures can be seen in Figure 2.23, with the annotation $Conv[(k,k),s,*d]$ where k equals the kernel size, s equals the stride and d equals the down sampling factor. If the down-sampling factor is smaller than one, for example 0.5, it is upsampling with a factor 2.



(a) Bottleneck residual structure.

(b) Inverted bottleneck residual structure.

Figure 2.23: Bottleneck residual structures from the *ResNet*.

2.5.5 Loss Functions

Loss functions are an important part of training the network. Loss is a metric describing the error between the network output $\hat{y}$ and the ground truth y. In the following section different loss functions used to train semantic segmentation networks are described.

Cross Entropy Loss

The Cross Entropy Loss (CEL) for class j is defined in Equation (2.15).

$$\mathrm{CEL}_j = -\sum_{i}^{n} (y_{ij} \log(\hat{y}_{ij}) + (1 - y_{ij}) \log(1 - \hat{y}_{ij})), \tag{2.15}$$

where n is the total amount of pixels per prediction, $\hat{y}_{ij} \in [0, 1], \mathbb{R}$ is the predicted label for pixel i and $y_{ij} \in \{0, 1\}$ is the true label for pixel i. $y_{ij} = 0$ if pixel i does not belong to class j and $y_{ij} = 1$ if pixel i belongs to class j.

The total CEL is then calculated by adding the class specific losses:

$$\mathrm{CEL} = \sum_{j} \mathrm{CEL}_j \tag{2.16}$$

One drawback of the cross entropy loss is that if one class constitutes a very small part of the image the loss for classifying it incorrect will not be large. Hence it will not have a major effect on how the network is updated towards. Since this is not a problem for balanced image classification tasks the cross entropy loss is very common for tasks with balanced datasets. But if the classes in a dataset are unbalanced, weighted losses for each class are necessary.

One possibility to overcome the unbalanced presence of classes is to use weights for the loss of each class:

$$\mathrm{CEL} = \sum_{j} W_j \mathrm{CEL}_j \tag{2.17}$$

The weights can be calculated as shown in Equation (2.18).

$$W_j = \frac{p_{\max}}{p_j} \tag{2.18}$$

Where $p_{\max}$ is the number of pixels in the class with the most pixels and p_j is the amount of pixels in the class j. Calculating the weights this way gives the most frequent class a weight of one and all the other weights are greater than one, such that each class is getting the same attention.

Dice Loss

Equation (2.19) shows the Dice loss for the class j where y_{ij} is the ground truth label for pixel i and $\hat{y}_{ij}$ is the prediction for pixel i [35]. The Dice loss function is the corresponding loss to the accuracy measurement IoU described in Section 3.4.1.

$$\mathrm{DL}_j = 1 - \frac{2\sum_{i}^{n} y_{ij}\hat{y}_{ij}}{\sum_{ij}^{n} y_{ij}^2 + \sum_{ij}^{n} \hat{y}_{ij}^2} \tag{2.19}$$

The authors of [35] state that the Dice loss function prevents the loss from getting stuck in a local minimum around the global minimum as the cross entropy loss function might do. This is often the case if the pixels are unbalanced, meaning that the network prioritizes large regions in the images such as road over smaller objects such as pedestrians. Hence using the Dice Loss helps to find smaller objects like trees or road signs in an image. They are likely to get neglected using un-weighted cross entropy loss, because the overall loss is low if the large prediction areas cover the ground truth areas.

3

Methods

In this chapter the methods used in this report are described in detail. First the datasets used to perform the training and evaluation are presented, and the pre-processing of the data is described. Then the used network architecture is presented along with the training setup. At the end of this chapter the evaluation of the results is described.

3.1 Datasets and Datahandling

There are real-world RGB-D datasets available for research in semantic segmentation. The dataset NYU2 [36], which contains images and depth maps from an indoor environment, was used a lot in the beginning of RGB-D semantic segmentation research. The NYU2 dataset contains labeled images for scenes like kitchens, living rooms and bedrooms. However since the purpose of this project is to perform semantic segmentation for autonomous vehicles, datasets from traffic environments are used. One such traffic environment RGB-D dataset is the Cityscape dataset [37]. The depth images in this dataset is calculated from stereo images and not obtained through depth scans as the NYU2 dataset. Moreover a dataset generated from the simulation environment Carla [27] is used.

3.1.1 CARLA Dataset

One way to generate data instead of using real-world data is to use the simulation environment Carla [27]. Carla has the advantage of providing auto-labeled semantic segmentation images and high quality depth images. Thus it has the ability to provide a large amount of data in short time. The Carla simulation environment contains several different city maps with scenery of suburbs, city centers, living areas as well as highways. There is a wide range of implemented sensors such as cameras with adjustable focal length and resolution and for our purpose, depth cameras. The depth images are processed as described in Section 2.3. The amount of pedestrians and cars is adjustable to simulate different traffic scenarios. The weather can be chosen between rain, sunshine and cloudy sky. For the semantic segmentation there are 13 classes available (see Table 3.1).

Table 3.1: Available classes in Carla [27].

Class	Label	[R,G,B] Color
Unlabeled	0	[0, 0, 0]
Building	1	[70, 70, 70]
Fence	2	[190, 153, 153]
Other	3	[250, 170, 160]
Pedestrian	4	[220, 20, 60]
Pole	5	[153, 153, 153]
Road line	6	[157, 234, 50]
Road	7	[128, 64, 128]
Sidewalk	8	[244, 35, 232]
Vegetation	9	[107, 142, 35]
Car	10	[0, 0, 142]
Wall	11	[102, 102, 156]
Traffic sign	12	[220, 220, 0]

Data Collection in Carla

The Carla dataset was generated in three of the seven available towns, to get a variation of data in suburb and city areas. We use *town02*, a small city with different scenery. Also *Town05* was selected, which is a squared-grid town with multiple lanes per direction and different living areas and lastly *town07*, a rural environment with narrow roads. For each used city about the same amount of images has been generated, to get a balanced dataset over all environments. To simulate people and traffic, about 150 persons and 100 cars were generated on each map that was used. This ensures that almost all images contain at least one car or pedestrian. Since lighting and weather conditions are correlating a lot with the performance of segmentation networks, the weather was chosen as sunny and cloudy afternoon, which equals the conditions of the Cityscapes dataset with good/medium weather conditions [37]. The resolution of the camera was chosen to (1024×512), since this was the highest resolution that could be used during training due to memory restrictions. An example of the used RGB and depth images can be seen in Figure 3.1.

Figure 3.1: Example of a RGB and depth image of a simulated traffic environment in Carla.

3.1.2 Cityscapes Dataset

The Cityscapes dataset contains 5000 densely annotated images from real-world RGB and depth data [37]. It was collected using a stereo camera mounted on a car. All data was collected during day time in good weather conditions. The camera captures stereo images and a disparity map is computed from these stereo images as shown in Section 2.3. The dataset was captured in over 50 cities in Germany which means that it provides a diverse set of scenery. The dataset has 30 classes of which 19 classes are used for evaluation. These classes are seen in Table 3.2. The images are captured in a resolution of (2048×1024) pixels. This enables the images to capture a lot of detail. Due to memory restrictions the images are downsampled to (1024×512) to use them for our network.

Table 3.2: Classes evaluated in the Cityscape dataset.

Class	Label	[R,G,B] Color
Road	0	[128, 64, 128]
Sidewalk	1	[244, 35, 232]
Building	2	[70, 70, 70]
Wall	3	[102, 102, 156]
Fence	4	[190, 153, 153]
Pole	5	[153, 153, 153]
Traffic light	6	[250, 270, 30]
Traffic sign	7	[220, 220, 0]
Vegetation	8	[107, 142, 35]
Terrain	9	[152, 251, 152]
Sky	10	[70, 130, 180]
Person	11	[220, 20, 60]
Rider	12	[255, 0, 0]
Car	13	[0, 0, 142]
Truck	14	[0, 0, 70]
Bus	15	[0, 60, 100]
Train	16	[0, 80, 100]
Motorcycle	17	[0, 0, 230]
Bicycle	18	[119, 11, 3]
Unlabled	255	[0, 0, 0]

The values of the camera for Equation (2.5), to calculate the depth images, used for the Cityscapes dataset are $b = 0.209$m and $f = 2262$mm. For remote objects the depth estimation is less accurate.

The disparity maps contain a lot of noise, hence so do the calculated depth images. The depth images are preprocessed with a blurring filter of size 5 to get rid of incorrect peak pixels in the depth image.

In order to evaluate how the training on Carla simulation data translates to real-world data, some of the classes of the Cityscapes dataset are merged to match the

classes of the CARLA dataset. A difference between Carla and Cityscapes is for example that the road lines are not separately labeled in Cityscapes. Table 3.3 shows how the classes are merged to the classes used in this project.

Table 3.3: Merging of classes from Cityscape classes to Carla classes.

Class in Cityscapes	Merged to
Bus	Car
Truck	Car
Bike	Car
Motorbike	Car
Rider	Pedestrian
Terrain	Vegetation
Parking	Road
Rail	Road
Guardrail	Fence
Bridge	Building
Sky	Unlabeled
Trafficlight	Traffic sign
Caravan	Car
Trailer	Car
Train	Car
Dynamic	Unlabeled
Ground	Other

3.1.3 Normalization of Data

The input data is normalized according to Equation (3.1). First it is scaled between 0 and 1. This is done by dividing the input by 255 since the input color space is from 0 to 255. After this it is normalized with a mean of $[0.485, 0.456, 0.406]$ and the standard deviations $[0.229, 0.224, 0.225]$ for the RGB channels, as it is recommended for *ImageNet* pretrained networks [34]. A mean of 0.450 and a standard deviation of 0.226 is used for the depth channel. The standard deviation for the depth channel was chosen since this means that most values will be in the range from -1 to 1. With this normalization the mean of each input channel is shifted to zero. The equation to perform this normalization is as follows:

$$normalized = \frac{(unnormalized - mean)}{standard\ deviation}. \tag{3.1}$$

3.2 Network Architecture

The network architecture is implemented using the Python library PyTorch [38] for DNN development. The framework enables computations on the GPU which yields faster training time compared to training on a CPU.

3.2.1 Network Design

A study in [39] shows that network types of the structure in Figure 2.17 (d) usually perform best in semantic segmentation tasks. Thus a structure similar to Figure 2.17 (d) and the *RedNet* introduced in Section 2.4 was used. This base was chosen, since it has a simpler structure than for example the *ACNet* and thus uses less computational power. The used network structure with context extraction layer and fusion module is shown in Figure 3.2. The base of our network is built on a *ResNet-50* encoder and a decoder with inverted bottlenecks containing deconvolution layers.

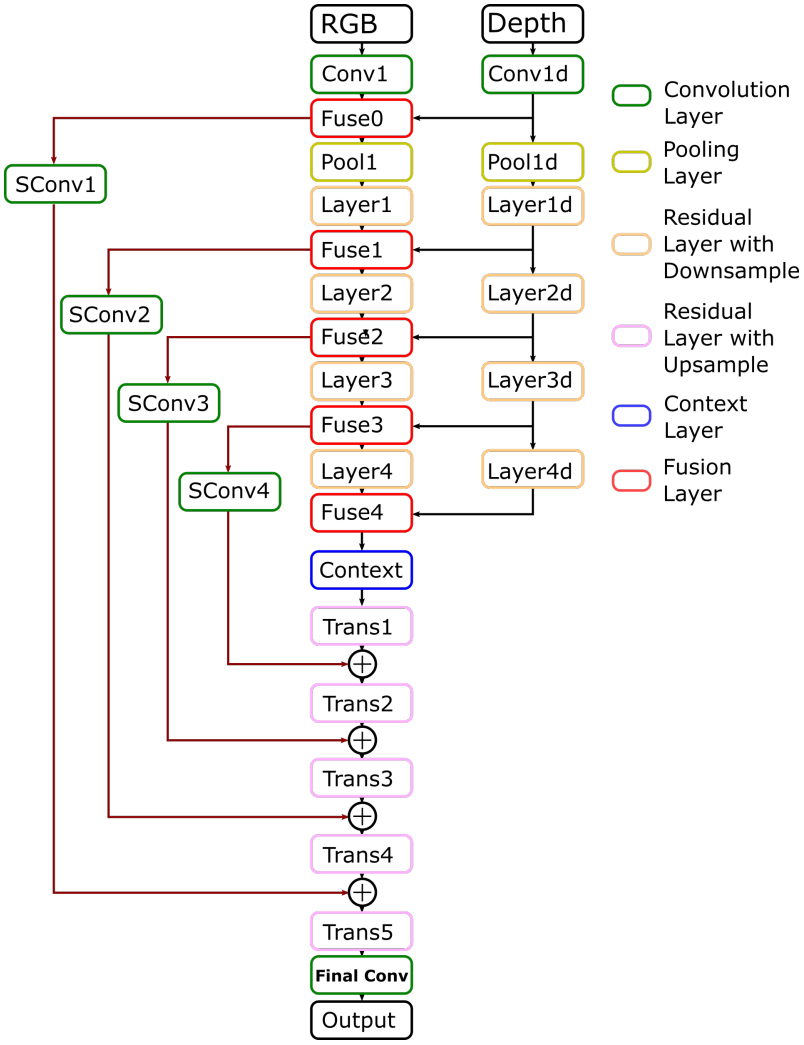


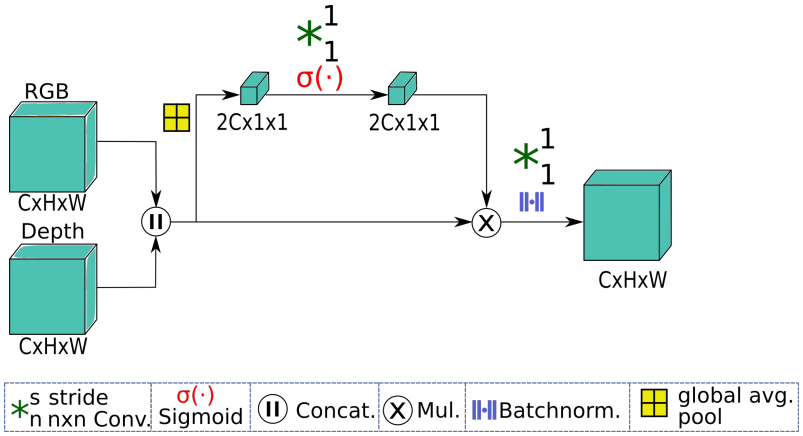
Figure 3.2: Network structure used in this project to perform RGB-D semantic segmentation. *DCF-Net 1* corresponds to an eASPP context layer, the *DCF-Net 2* corresponds to a WASP context layer.

The *ResNet-50* encoder was chosen since it is available with pre-trained weights on the *ImageNet* dataset, which contains 1000 classes for pure classification of images [40]. Furthermore due to its residual structure our approach is converging faster during optimization compared to the *BiSeNet*. The output of the *ResNet-50* encoder is a feature map with rich features i.e. complex features. The bottleneck structure and the inverted bottlenecks in the decoder, are described in Section 2.5.4.

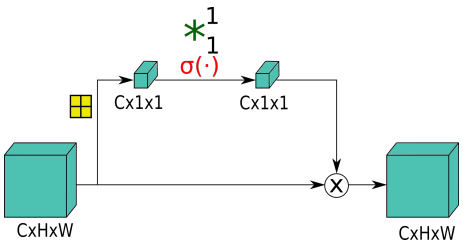
To improve the accuracy of the network base, it is extend with a fusion module to fuse the depth information continuously into the RGB branch. Continuous fusion is more effective and achieves higher accuracy than early fusion with depth-aware convolutions or late fusion with separate encoders [32], [13], [29], [39]. The fusion was done in an attention oriented manner. To extract the context information of the feature maps more effectively, our approach is also extended to two versions using either the eASPP or the WASP layer described in Section 2.1.5.

3.2.2 Fusion Module

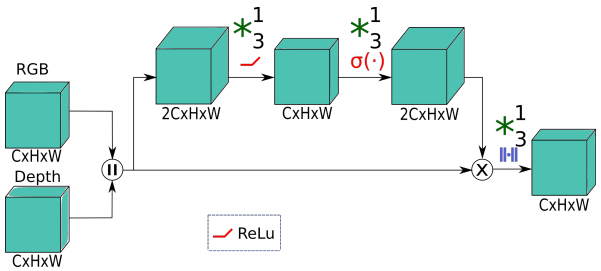
To create a computationally effective fusion module an attention sort mechanism is used. Therefore the RGB and the depth feature maps are concatenated. The attention value for each channel is generated by pooling each channel down to (1×1) channels using global average pooling. A (1×1) convolution is applied to get channel correlation between the channels. Then a sigmoid activation function is used to normalize the attention weights between 0 and 1. To weight the channels by its importance each channel is multiplied with its corresponding attention value. Since there are still twice as many channels as originally, the weighted channels are convoluted again with a (1×1) filter down to the original input size of each RGB and depth feature map. The proposed fusion module can be seen in Figure 3.3 (a).



(a) Our fusion module for RGB and depth feature maps.



(b) Attention Module from *ACNet* [31].



(c) Fusion module from *AdaptNet++* [14].

Figure 3.3: Different fusion and attention modules.

The fusion module design is based on the attention complementary module of the

ACNet [31] (Figure 3.3 (b)), to create the attention values for each channel. The structure to fuse RGB and depth streams, was adapted from the *AdaptNet++* (Figure 3.3 (c)). It concatenates the RGB and depth feature maps, and convolutes it down after an attention multiplication to the wanted number of channels. Because the way the *AdaptNet++* convolutes and computes the attention values is computational more expensive, we adapted the attention branch from the *ACNet* instead.

3.2.3 Context Information

The layer with the highest context information density is the output of the encoder. In our case with the *ResNet-50* encoder, its size is 32 times smaller than the input size. To extract all context information and relations of this feature map, we use the eASPP layer proposed in the *AdaptNet++* network [14] as well as the WASP structure (see Figure 2.11 and 2.12) to get a comparison of their effectiveness. With the several dilated convolutions on different rates, these layers are able to extract the connection between the features all over the image. The original ASPP layer is not used, since it requires more computations producing the same accuracy as the eASPP.

3.2.4 Spatial Information

Spatial information is information about the position of an object in an image. This information is captured and kept with shortcuts between the encoder and decoder structure. When the encoder decreases the size of the image to rich feature maps the location, i.e. spatial information of the objects in the image are lost. The shortcuts are used to retrieve this information by passing the higher resolution, low level feature maps to the decoder. Thus the spatial information is extracted, this is crucial for high resolution scene understanding . Each shortcut is using a (1×1) convolution to extract the correlation between the pixels of the feature maps.

3.2.5 Final Network Architectures

To evaluate the effect of the usage of fusion modules and different context extraction layers three networks are implemented. The *Depth-Context Network* (*DC-Net*) is the adaption of the base structure Figure 2.18, with an eASPP layer for context extraction but without fusion modules. The *Depth-Context-Fusion Network 1* (*DCF-Net 1*) is the base structure with an eASPP layer. To be able to evaluate the effect of the designed fusion module the *DC-Net* and the *DCF-Net 1* are compared. And the *Depth-Context-Fusion Network 2* (*DCF-Net 2*) is the base structure with the WASP layer for context extraction. To compare the eASPP and WASP layer, the *DCF-Net 1* and *DCF-Net 2* are compared.

3.3 Training

Training of the network is done on a desktop computer with 32 GB of RAM and a Nvidia RTX 2080 Ti GPU with 11 GB of dedicated graphics memory [41]. This enables the use of the graphics card for the tensor calculations which makes the training faster compared to training on a CPU. We use the method of early stopping to reduce optimization time and avoid overfitting, where the training runs for at least a set amount of epochs and then until the validation loss converges to a threshold area [42]. The threshold area in our case was chosen to 0.0005 for both used loss functions. If the loss stays for five epochs in this area the training is terminated.

The dataset with Carla images contains 1360 images for training and validation and 405 images for testing. From the Cityscapes dataset 2975 images are used for training and validation and 500 used for testing. The split ratio for train and validation images was chosen as 10% for both datasets.

3.3.1 Loss Functions

After some initial tests using only Dice loss or only CEL it turned out that using only Dice loss does not converge at all. Using only CEL resulted in decent results. Nevertheless the best method is to split the training of the network into two parts.

During the first part of training, cross entropy loss with weights is used since this loss function keeps the gradient for each pixel, thus it looks for the correlation between input, network weights and the pixel output. The first part of training is run until the pixel accuracy has converged. The weights used for the CEL are initially calculated using Equation (2.18). The calculated weights for the Carla dataset can be seen in Table 3.4.

During the second part of training the Dice loss function is used since it is the loss function corresponding to the mean Intersection over Union (mIoU) and it increases the mIoU by a significant margin. The mIoU is described more in Section 3.4.1.

3.3.2 Optimizer

The networks are optimized with the SGD algorithm described in Section 2.5.1, using momentum and a momentum weight factor of $\gamma = 0.8$. The learning rate is multiplied by 0.8 every 10^{th} epoch.

Table 3.4: Weights for the CEL function. Each weight stands for the value used to scale the loss of the corresponding class.

Class	Weights
Unlabeled	1
Building	1.9
Fence	30.5
Other	25.2
Pedestrian	85.9
Pole	40.2
Road line	80.2
Road	1.1
Sidewalk	3.3
Vegetation	5.9
Car	8.9
Wall	57.2
Traffic sign	250.3

3.4 Evaluation

To perform real-time semantic segmentation there are two important evaluation properties, the runtime and accuracy of the predictions. With these evaluation properties the resolution of the images and the relevance of the artificial data are also evaluated. Lastly the relevance of depth data to robustness is investigated.

3.4.1 Evaluation of Accuracy and Runtime

The accuracy of the predictions is measured in two different ways. The first way is mean pixel accuracy (pAcc) that is defined in Equation (3.2).

$$\text{pAcc} = \frac{\text{Correctly labeled pixels}}{\text{Total number of pixels}} \tag{3.2}$$

One drawback using mean pixel accuracy (pAcc) is, that poor performance on classes that constitute a small part of the image does not affect the mean pixel accuracy a lot. For example an image with two classes, sea and boat, where sea constitutes 95% of the image and boat the other 5%. If the network labels the entire image as sea it will reach a mean pixel accuracy of 95% which is high but neglecting the smaller class.

Therefore a second way to measure the accuracy is used, the Intersection over Union (IoU) and the mean Intersection over Union (mIoU). A graphical illustration of the area of intersection and the area of union can be seen in Figure 3.4.

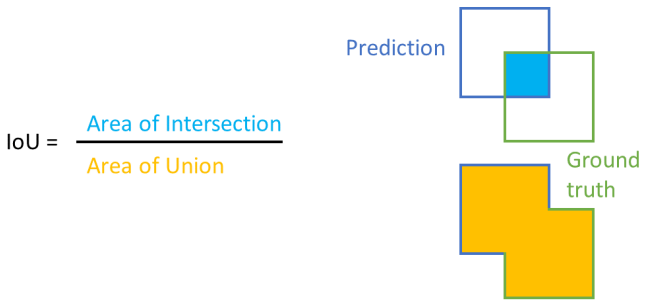


Figure 3.4: Visualization of the Intersection over Union (IoU).

The mIoU is defined in the following equation.

$$\text{mIoU} = \frac{\sum_{\text{classes}} \frac{\text{Area of intersection}}{\text{Area of union}}}{\text{no. classes}} \tag{3.3}$$

By measuring the accuracy this way incorrect labeling of small classes has a larger effect on the accuracy value. For the previous example with the boat and the sea the resulting mIoU is 47.5% which reflects the accuracy of the prediction in a more nuanced manner [43].

To evaluate the runtime T for one image, or the Frames per Second (fps), the time is measured in nanoseconds before performing a prediction for n images, as well as saving the prediction image and again after. These measures are defined as:

$$\text{fps} = \frac{n}{t_1 - t_0} \tag{3.4}$$

$$T = \frac{1}{\text{fps}} \tag{3.5}$$

Where t_0 is the measured time in seconds before the prediction and t_1 is the measured time after the prediction of n images.

During the performance evaluation no gradients are used to improve the performance, since they are only needed for training. All GPU kernels are synchronized such that the time measurement is waiting for all GPU kernels to finish. To filter out uncertainty, batches of 20 RGB-D images are processed 50 times, resulting in 1000 image predictions.

3.4.2 Evaluation of Resolution

The resolution of the images is an important factor for the network performance. If the resolution is too low the network has trouble finding some features or might classify them incorrectly. If the resolution is too high the images might use more space and processing power than necessary without adding any additional information to the segmentation process. It might also make the runtime performance of the network significantly slower. To evaluate the effect of the resolution the same network will be trained for different image sizes. The larger resolution for our comparison is (1024×512) and the smaller is (512×256).

3.4.3 Evaluation of the Relevance of Artificial Data

Carla generates labeled data as described in Section 3.1.1. It is however of great importance to know how well a network trained on artificial data performs in the real-world. This is evaluated by training a network on the Carla dataset and then evaluating it using the Cityscapes dataset mentioned in Section 3.1.2.

3.4.4 Evaluation of the Relevance of Depth Data

Using depth data in the segmentation of images has the potential to improve the quality of the segmentation. Especially if there for some reason are disturbances in the RGB and or depth image due to light conditions or fog for example. To evaluate the role of the depth data in making the segmentation more robust an adaptation of the *DCF-Net 2* that only uses RGB images was made, called *RedGreenBlue-Context Network* (*RGBC-Net*). The accuracy of the segmentation was evaluated on both the *DCF-Net 2* and the *RGBC-Net*. The weights in the filter for Gaussian blur is defined in the equation below.

$$G(p_{i,j}) = \frac{1}{2\pi\sigma^2} \mathrm{e}^{-\frac{i^2+j^2}{2\sigma^2}}, \tag{3.6}$$

were i and j are the distance of pixel p from the center of the filter and σ is the standard deviation. The Gaussian blur was added to the RGB images and the evaluation was done again, no blur was however added to the depth image. This represents disturbances in the RGB image due to for example dust on the camera lens, which is a disturbance that does not affect the depth image. Then blur was added to the depth image and lastly to both the depth and RGB image. An example of a blurred image, using a Gaussian blur filter of size 15 with $\sigma = 3$, can be seen in Figure 3.5.

Figure 3.5: Visualization of disturbance simulation using Gaussian blur.

4

Results

In this chapter the results from the training and testing of the networks are presented. The chapter is divided into six sections, where the first deals with accuracy and runtime aspects of the networks. Sections two and three present the evaluations of data resolution and the relevance of artifical data. The last section presents the relevance of depth data on robustness.

4.1 Evaluation of Accuracy and Runtime

In this section the *DC-Net*, *DCF-Net 1* and *DCF-Net 2* are evaluated to compare the effect of different context extraction layers and the fusion module on accuracy and runtime performance. A detailed description of the networks can be seen in Section 3.2.1. Training is done using a learning rate decay of 20% every 10^{th} epoch. The networks are trained on 1360 Carla images with a resolution of (1024×512). It is not possible to load all images at once due to memory restrictions, hence the networks are trained in multiple sessions of 800 images. One epoch does in this case refer to training one time over all 800 images used in this session and not over all 1360 images used for training. The used loss functions and initial learning rates for each session are shown in Table 4.1.

Table 4.1: Learning parameters and loss functions for training sessions.

Session	Epochs	Loss function	Initial learning rate
0	10	CEL	0.1
1	20	CEL	0.01
2	20	CEL	0.001
3	20	CEL	0.0005
4	40	CEL	0.0001
5	40	Dice loss	0.0001

First the CEL is used during training with a high learning rate to converge the parameters to an area around the optimum. The learning rate is then decreased in each session to ensure convergence to the true optimum of the loss function. The loss and pixel accuracy progress when training the *DCF-Net 2* with CEL is seen in Figure 4.1. For the last training session the Dice loss is used to increase the mIoU accuracy.

The loss and pixel accuracy of the last two training sessions for the different network approaches can be seen in Figure 4.2 and 4.3. The final loss for training can be seen in Table 4.2. The *DCF-Net 2* has the highest training pixel accuracy and lowest loss for both loss functions.

Table 4.2: Loss for the networks after training sessions with different loss functions. CEL after session four, Dice loss after session five.

Network	CEL	Dice loss
DC-Net	0.156	0.043
DCF-Net 1	0.141	0.042
DCF-Net 2	0.107	0.032

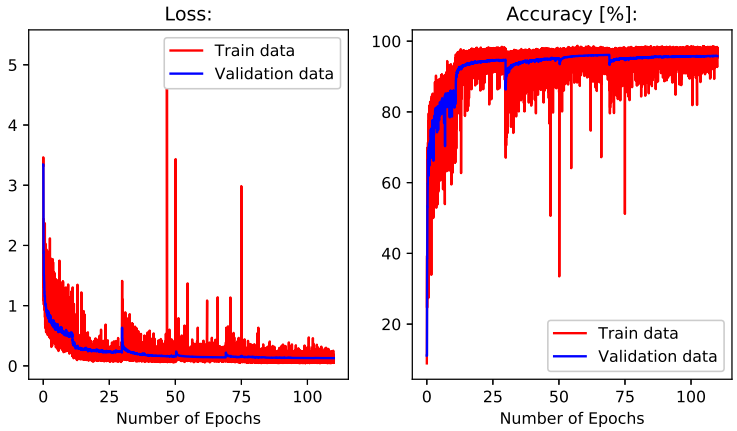


Figure 4.1: CEL and pixel accuracy progress during training session 0-4 for the *DCF-Net 2* on the Carla dataset.

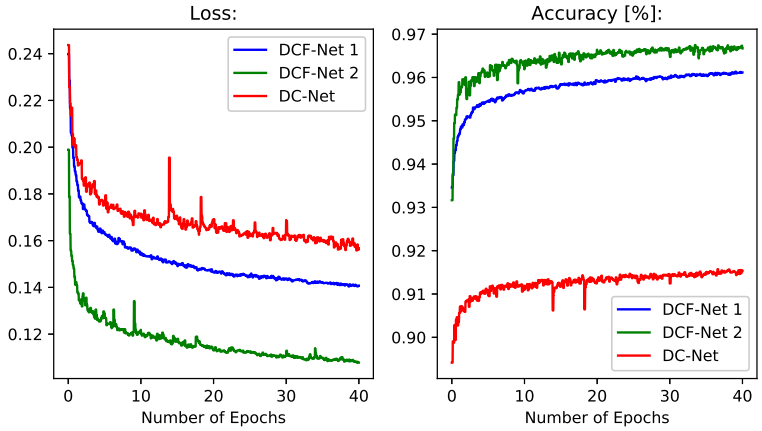


Figure 4.2: CEL and pixel accuracy comparison between the different networks for training session 4.

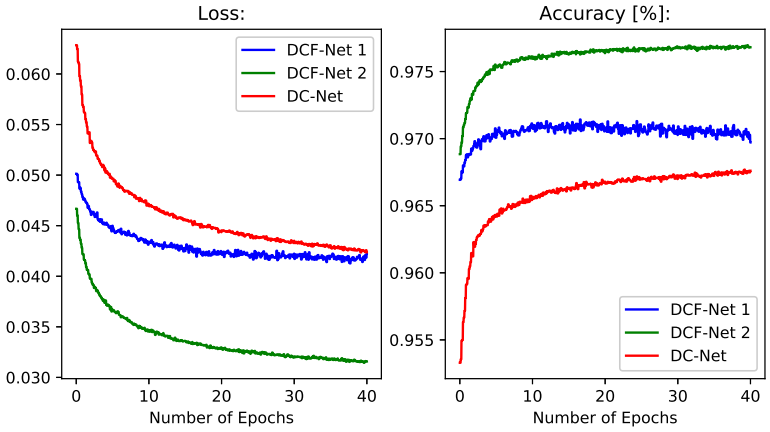


Figure 4.3: Dice loss and pixel accuracy comparison between the different networks for training session 5.

The networks are tested on a test set containing 405 Carla images. The accuracy and runtime on the test set is shown in Table 4.3 below.

Table 4.3: Final results testing the trained networks on the Carla dataset.

Network	**Pixel accuracy**	**mIoU**	**fps**	**Runtime**
DC-Net	96.7%	70.5%	15.9	62.9 ms
DCF-Net 1	96.9%	71.5%	14.2	70.4 ms
DCF-Net 2	96.9%	71.8%	15.4	64.9 ms

Comparing the runtimes it can be seen that the *DC-Net* without fusion modules is 7.5ms faster than the *DCF-Net 1*. This is a difference of 10% between the *DC-Net* and the *DCF-Net 1*. Furthermore the *DCF-Net 2* compared to the *DCF-Net 1* is 5.5ms or 7.8% per image faster.

A common issue for the networks is that they struggle with the diverse class *other* containing objects that do not belong to any other classes. Furthermore thin structures such as fences and poles further away in the image also have a low accuracy. These issues can be seen in the overview of the IoUs for each class shown in Figure 4.4. Tables with exact IoUs can be seen in Table B.1, B.2, and B.3. The classes *fence* and *other* have the lowest IoU. The IoU for *fence* is 35% using the *DC-Net* and 40% using the *DCF-Net 2*. For *other* the IoU is 37% using the *DC-Net* and 44% using the *DCF-Net 2*. Thus these class IoUs are low compared to the other classes.

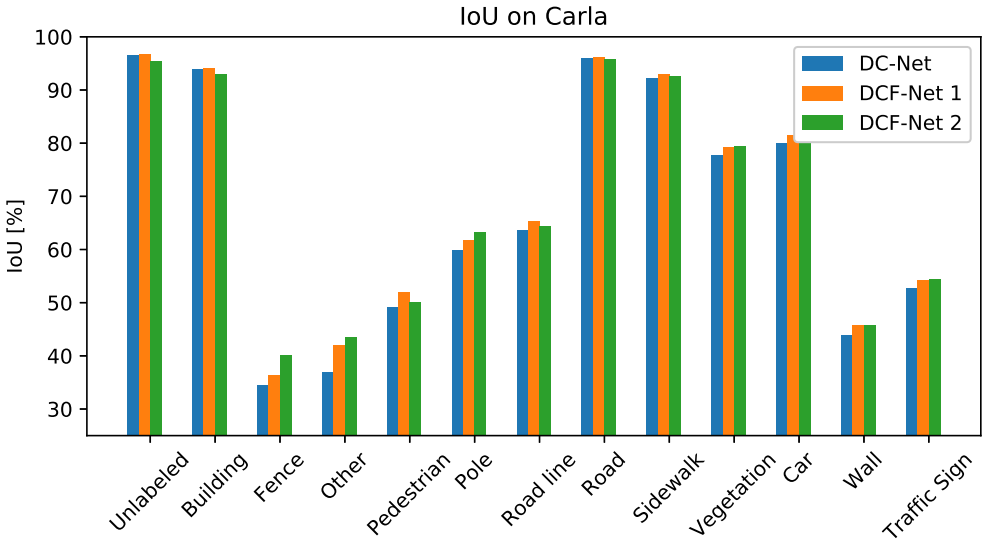


Figure 4.4: Class IoU on the Carla dataset for the different networks.

In Figure 4.5 and 4.7 example inputs to the networks are shown with the corresponding ground truth and predictions in Figure 4.6 and 4.8.

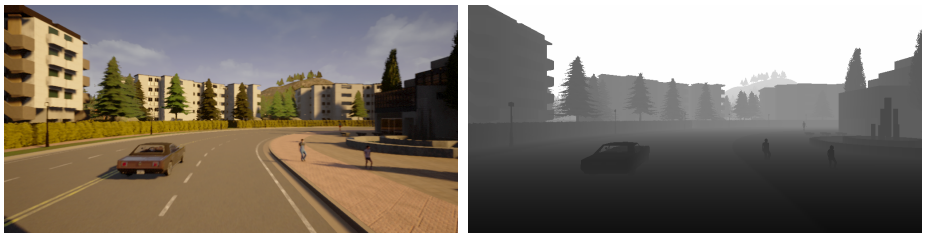


Figure 4.5: RGB and depth image for corresponding segmentations in Figure 4.6.

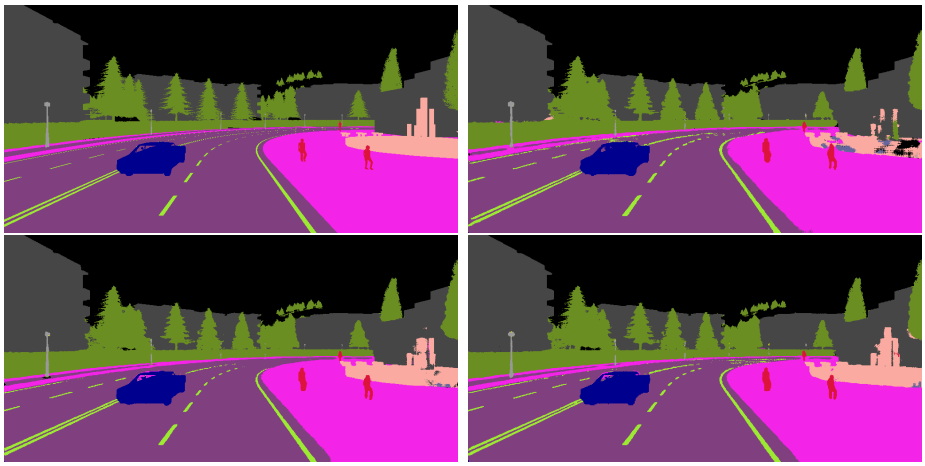


Figure 4.6: Ground truth (top left), *DC-Net* (97%pAcc, 76%mIoU) (top right), *DCF-Net 1* (98%pAcc, 80%mIoU) (bottom left), *DCF-Net 2* (98%pAcc, 81%mIoU) (bottom right).

Figure 4.7: RGB and depth image for corresponding segmentations in Figure 4.8.

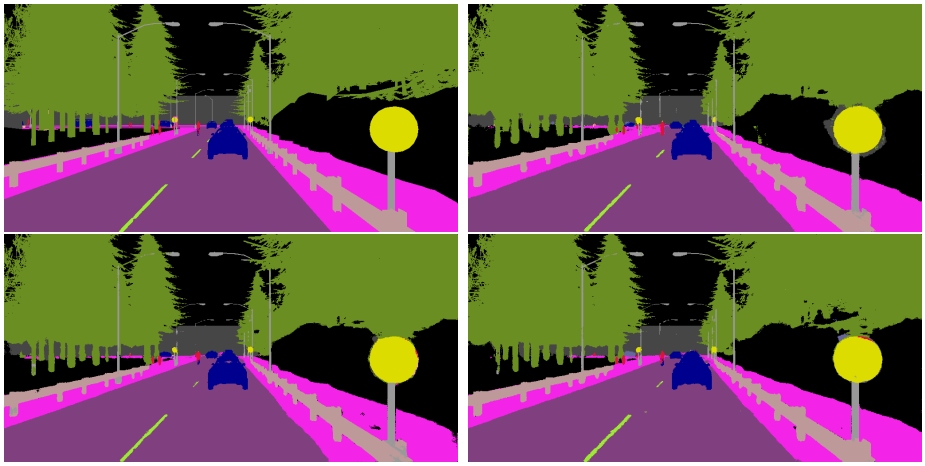


Figure 4.8: Ground truth (top left), *DC-Net* (95%pAcc, 74%mIoU) (top right), *DCF-Net 1* (96%pAcc, 75%mIoU) (bottom left), *DCF-Net 2* (97%pAcc, 77%mIoU) (bottom right).

4.2 Evaluation of Resolution

Since the *DCF-Net 2* has the highest mIoU on the Carla dataset and has a higher frame rate, i.e. more fps than the *DCF-Net 1*, it is used for the resolution comparison.

To evaluate the effect of the image resolution on the segmentation results, the *DCF-Net 2* was trained on both high resolution (1024×512) and low resolution (512×256) images. The comparison of the test results can be seen in Table 4.4. Even though the pixel accuracy differs by only 1.2 percentage points, the difference in mIoU is 6.7 percentage points. The higher resolution results in a longer runtime and therefore in a lower frame rate. A plot of the IoU for all classes for the low and high resolution images is shown in Figure 4.9. The table with the corresponding values can be seen in Table B.3 and B.4. In all classes containing fine structures such as *pole*, *pedestrian*, *road line* and *traffic sign* the iou of the high resolution images is significantly higher than for the low resolution. For structures covering larger areas of the image like *building*, *road* and *unlabeled* the difference is minor.

Table 4.4: Evaluation results of the *DCF-Net 2* using high and low resolution data.

Resolution	Pixel Accuracy	mIoU	fps
512x256	95.7%	65.1%	39
1024x512	96.9%	71.8%	15.4

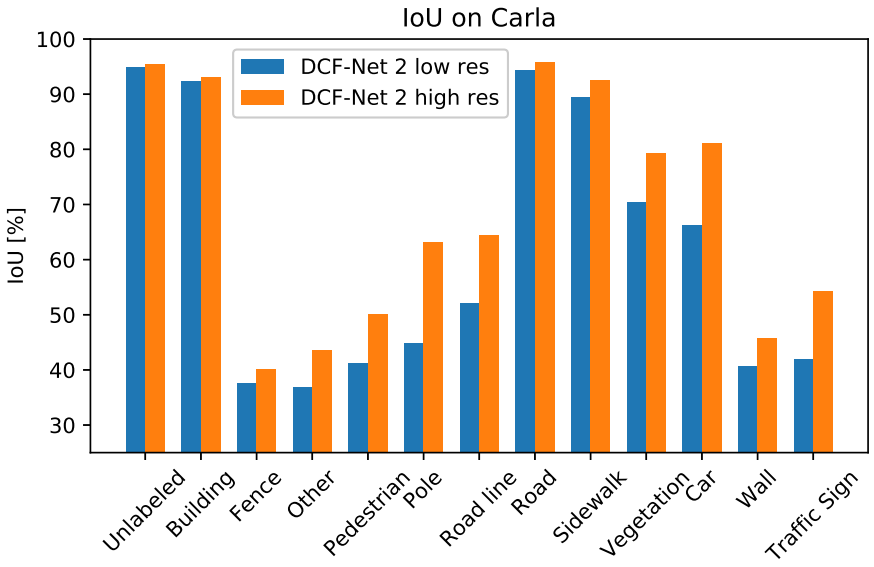


Figure 4.9: IoU comparison for the *DCF-Net 2* using high and low resolution data.

4.3 Evaluation of the Relevance of Artificial Data

The evaluation of the relevance of artificial data was done by using the *DCF-Net 2* trained only on the CARLA dataset and evaluating it on the Cityscapes dataset. The *DCF-Net 2* is chosen since it has the highest accuracy on the Carla dataset and has a faster runtime than the *DCF-Net 1*. The result can be seen in Table 4.5.

Table 4.5: Evaluation of the *DCF-Net 2* trained only on the Carla dataset performing segmentation on the Cityscapes dataset.

Network	Pixel accuracy	mIoU
DCF-Net 2	30.5%	11.8%

A visual example of the evaluation can be seen in Figure 4.10. The network finds a lot of edges and boundaries of objects correctly, but it does classify objects incorrectly, e.g. the car that is classified as *other*. In general the features are captured, but they need to be fused to the right context information to result in correct classification. Road lines are not a separate class in the Cityscapes dataset but they are in the Carla dataset. Hence the network captures road lines but they are not present in the ground truth.

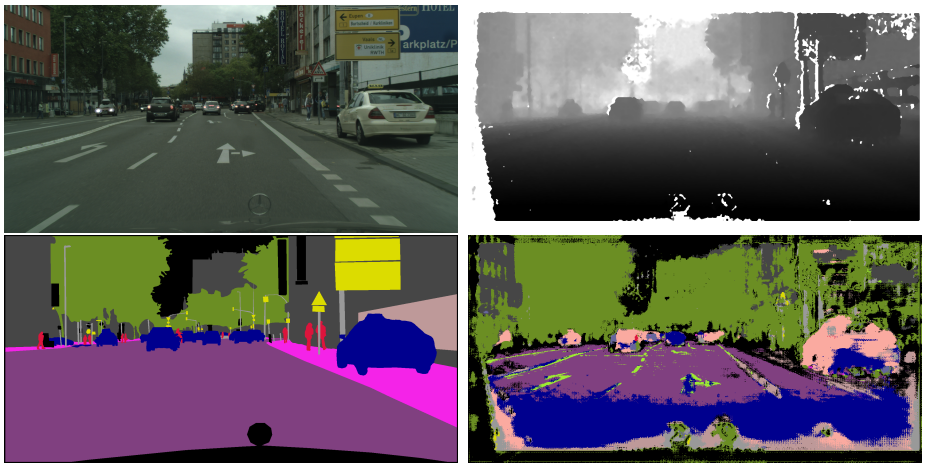


Figure 4.10: Example of the *DCF-Net 2* trained only on the Carla dataset performing segmentation on Cityscapes dataset. RGB input (top left), depth input (top right), ground truth (bottom left) and segmented output (37%pAcc, 13%mIoU) (bottom right).

4.3.1 Training on Cityscapes

Even though the *DCF-Net 2*, only trained on the Carla dataset, classifies objects incorrectly, it is able to detect structures and features in the Cityscape images. Hence transfer-learning was used to adapt the network to the new dataset. The results of 40 further epochs CEL and 40 epochs Dice loss training can be seen in Table 4.6. To evaluate whether transfer-learning is useful, a *DCF-Net 2* was trained only using Cityscapes data according to the training procedure in Table 4.1. The results of this training for comparison can be seen in Table 4.6 as well the corresponding IoU values in Figure 4.11.

Table 4.6: Final results after training the *DCF-Net 2* using CEL followed by Dice loss and testing on the Cityscapes dataset. One version with transfer-learning (TL) from Carla and one version using only Cityscapes data (CS) during training.

Network	Pixel Accuracy	mIoU
DCF-Net 2 (TL)	93.4%	61.7%
DCF-Net 2 (CS)	93.0%	68.4%

An example prediction of the test set can be seen in Figure 4.12. The table with the corresponding values can be seen in Table B.5 and B.6. The Carla classes *other* and *road line* are not evaluated since they are not present as separate classes in the Cityscape dataset. Furthermore, the class *wall* is not evaluated since there are only very few images with *wall* in the dataset.

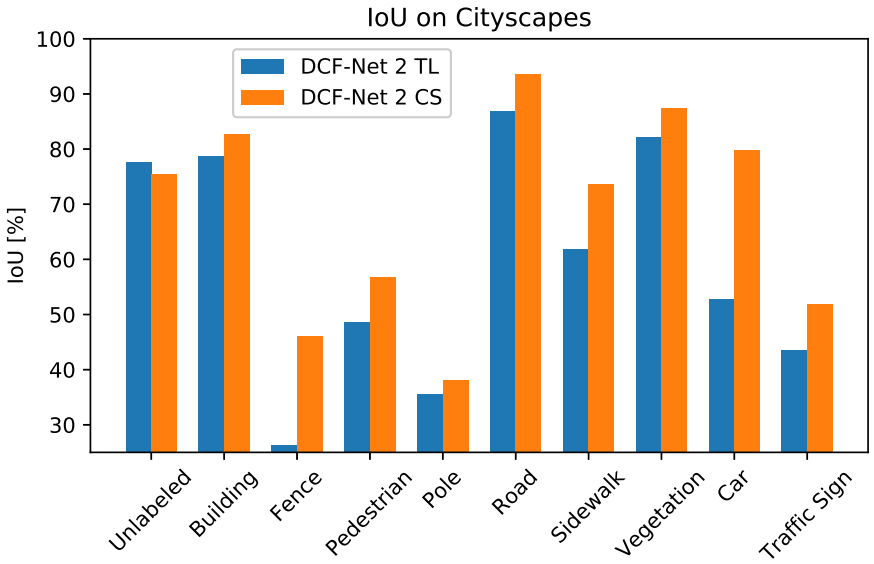


Figure 4.11: IoU for the *DCF-Net 2* on Cityscapes dataset trained using transfer-learning (TL) and using only Cityscape data (CS).

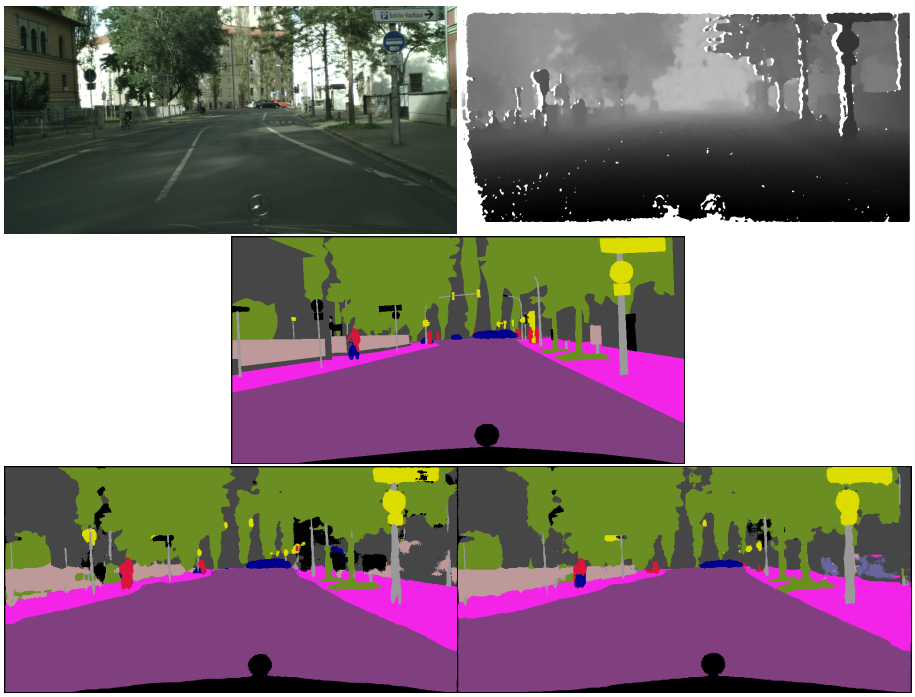


Figure 4.12: RGB (top left), depth (top right), ground truth (center), segmented output (TL) (90% pAcc, 65% mIoU) (bottom left) and segmented output (CS) (91% pAcc, 66% mIoU) (bottom right). Were the network trained using transfer-learning is denoted (TL) and using only Cityscape data is denoted (CS).

4.4 Evaluation of the Relevance of Depth Data

To evaluate the relevance of depth data a comparison network, *RGBC-Net* using only RGB data was created. It has the same structure as the *DCF-Net 2* except that it does not have a depth branch. A comparison of the difference between the pixel accuracy and mIoUs of the *RGBC-Net* and *DCF-Net 2* trained on the Carla dataset can be seen in Table 4.7. Both *DCF-Net 2* from transfer-learning (TL) and training only on Cityscapes (CS) are evaluated in the robustness study and compared to the *RGBC-Net* for the Cityscapes dataset as shown in Table 4.8. The check-marks(✓) and crosses(✗) show if an image has been blurred or not. The Not Applicable (N/A) for the depth image of the *RGBC-Net* is used since the *RGBC-Net* does not use depth images. The Gaussian blur as defined in Equation 3.6, is added to the images using $\sigma = 3$ and a filter size of 15. While the difference in mIoU with no blurred images between the *DCF-Net 2* and *RGBC-Net* on Carla is 11.4%, the difference on the Cityscapes dataset is only 2.6% for the *DCF-Net 2* (TL) and 9.3% for the *DCF-Net 2* (CS). Especially for classes with thin structures like *pole* the use of depth images increase the IoU significantly by 33% on the Carla dataset, whereas the difference for *building* is only 8% as can be seen in Figure 4.13.

Table 4.7: Accuracy of networks using data with and without Gaussian blur for the Carla dataset.

Network	Pixel accuracy	mIoU	RGB blur	Depth blur
DCF-Net 2	96.9%	71.8%	✗	✗
RGBC-Net	92.2%	60.4%	✗	N/A
DCF-Net 2	94.1%	61.3%	✓	✗
DCF-Net 2	86.5%	45.9%	✗	✓
DCF-Net 2	82.8%	37.2%	✓	✓
RGBC-Net	72.3%	31.4%	✓	N/A

In Table 4.8 for the Cityscapes dataset, it can be seen that the *DCF-Net 2* (TL) is only losing 15.8% accuracy in mIoU when only the RGB image is blurred, whereas the *DCF-Net 2* (CS) and *RGBC-Net* drop to 23.1%mIoU and 21.3%mIoU respectively. On the other hand, the *DCF-Net 2* (CS) is with 66.1%mIoU, 15.5% higher in accuracy than the *DCF-Net 2* when only the depth image is blurred. When both images are blurred the *DCF-Net 2* (TL) is again outperforming the other networks. With 37.6%mIoU it has a significantly higher mIoU than the *DCF-Net 2* (CS) and the *RGBC-Net*.

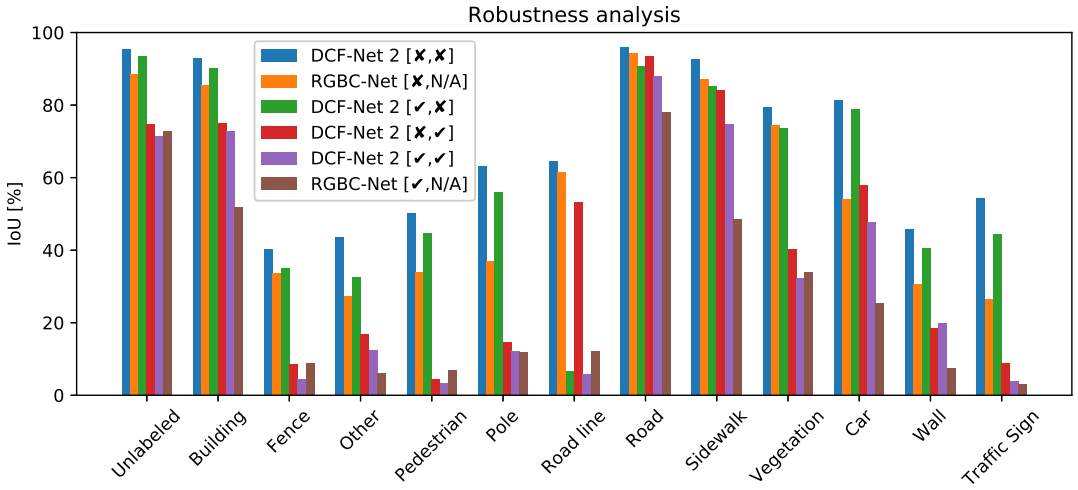


Figure 4.13: Comparison of IoU for different blur combinations using the *DCF-Net 2* and *RGBC-Net* on the Carla dataset. [✓, ✓], the first check-mark shows if the RGB image is blurred and the second if the depth image is blurred.

Table 4.8: Accuracy of networks using data with and without Gaussian blur for the Cityscapes dataset. Again (TL) denotes the *DCF-Net 2* from transfer-learning and (CS) denotes the *DCF-Net 2* from training only on Cityscapes data.

Network	Pixel accuracy	mIoU	RGB blur	Depth blur
DCF-Net 2 (TL)	88.0%	61.7%	✗	✗
DCF-Net 2 (CS)	93.0%	68.4%	✗	✗
RGBC-Net	87.9%	59.1%	✗	N/A
DCF-Net 2 (TL)	80.9%	45.9%	✓	✗
DCF-Net 2 (CS)	49.8%	23.1%	✓	✗
DCF-Net 2 (TL)	78.8%	50.6%	✗	✓
DCF-Net 2 (CS)	91.9%	66.1%	✗	✓
DCF-Net 2 (TL)	72.8%	37.6%	✓	✓
DCF-Net 2 (CS)	43.2%	17.6%	✓	✓
RGBC-Net	46.8%	21.3%	✓	N/A

5

Discussion

In this chapter the results are discussed with regards to the research questions and other relevant aspects. The first section deals with questions concerning accuracy of the different approaches. Section two discusses resulting runtime and how different factors affect it. The third section discusses the effect data resolution has on the accuracy. In section four the ability to use artificial data to train a network is discussed. Furthermore the results of the transfer-learning on Cityscapes data and the relevance of depth data are discussed. The last section discusses the results of this project compared to previous works.

5.1 Evaluation of Accuracy

In Figure 4.4 it can be seen that the IoU for the class *fence* is only at about 40% for all networks. These results are due to its thin pattern structure that is often labeled as a solid object and hence the "over labeling" is decreasing the mIoU. A car usually has a large coherent area which affects the feature maps in the encoder more, resulting in more information about the car at the end of the encoder. A fence on the other hand usually covers a small and non coherent area and is thus mostly recognized through the spatial information feature maps fed to the decoder through shortcuts. The *Pedestrian* class has a IoU of 60%, just as the *fence* it is also below the average mIoU of 71.8%. A reason for this is the position of persons, which constitute the *pedestrian* class. Persons can be either on the street, on the sidewalk or sitting on a motorcycle. Cars on the other hand are usually on the road and trees are usually in a green tone and never on the road. Thus the context information for a car, which is always on the road is easier to learn than for a person, which can occur in different places in the image.

In Figure 4.1 drops in pixel accuracy and increases in loss can be seen at the beginning of new sessions. This is due to the change in images used for training, hence the network is not used to these image. These drops and increases gets smaller as the training progresses since the network has now seen all the images.

5.1.1 Effects of the fusion module

To compare the effect of the fusion module the relevant numbers are the results of the *DC-Net* and the *DCF-Net 1* seen in Table 4.3 and Figure 4.4. They are the same network except for the fusion module that is added to the *DCF-Net 1*. For the overall result it is only a small difference of 1% in the mIoU value. This can be explained by the structure of the network and the position of the fusion module in it. The fusion module is fusing the feature maps after each layer of the *ResNet-50* encoder, such that the skip connection contains both RGB and depth information. In each layer each encoder of the RGB and depth branch extract the same features, thus simple addition is already fusing the information of both branches in a reasonable way. Nevertheless, the *DCF-Net 1* is achieving a higher IoU when it comes to classes that contain fine structures such as *fences*, *poles*, *pedestrians* and *road lines*. Here the fusion module is improving the IoU by putting a higher attention to the important channels. Fine structures, standing detached, contain higher gradients in the depth images of the Carla dataset and thus more information about their position. This results in a shift of the mIoU from 49% on *pedestrian* and 60% on *pole* without the fusion module to 52% on *pedestrian* and 62% on *pole* with the fusion module. This effect does not apply to classes like *building* and *car*, here the network without fusion module is performing equal or slightly better than with a fusion module.

These higher accuracy values come with a trade-off to runtime. Here the *DC-Net* is 1.7 fps faster than the *DCF-Net 1*. This longer runtime occurs due to the additional computations that are done in the fusion module.

5.1.2 Effect of Carla annotation

The way Carla annotates classes to objects are in some aspects hard to interpret for the network. For example when annotating a car where it is possible to see through the windows, the car body will be annotated as *car* and what is seen through the windows will be annotated as that class. In one way this seem logical since windows are see through. But it means there can be a part of a road, a building or a tree inside the car structure. Hence the context information from the car roof and body can not be used to predict that it is for example a building behind the window. Annotating the images this way could thus make it harder for the network to segment things correctly.

In Cityscapes cars are labeled entirely as *car*, neglecting the objects seen through its windows [37]. It is however beneficial for an autonomous vehicle to be able to see through another cars windows, even if it is more complicated than just classifying the entire car as car. For example, if the autonomous vehicle is driving down a street with parked cars on the sides and a pedestrian enters the street from behind a car. If the autonomous car can spot the pedestrian before it enters the road the autonomous car can break in time and not hit the pedestrian.

Another difference between the Carla and the Cityscapes annotation standard is that the back of traffic signs in Carla are marked as *traffic sign* while in Cityscapes

they are marked as *unlabeled*. This means that when the network is trained on Cityscapes it learns the difference between a sign with useful information and the backside of the sign without any information.

5.1.3 Effect of Loss Function

As seen in Figure 5.1 the different loss functions affect the training result. The main difference in the segmentation result can be seen around the traffic on the left side of the image. The segmentation by the network only trained on CEL has a lot of incorrect segmentation around the traffic sign while the network fine tuned using the Dice loss makes a better segmentation in this area. This is due to the fact that the Dice loss weighs all classes the same without considering the occurrence of a class. Low accuracy on a small class is given the same amount of attention as low accuracy on a large class. The loss for each class can take values between zero and one. Another aspect of the dice loss function is that it punishes to large predictions hard. If the prediction $\hat{y}$ predicts all of y but also predicts incorrectly that an additional area also belongs to y, the dice loss will punish this hard since the $\hat{y}$ is squared in the denominator, see Equation (2.19). And since if a large part of the prediction is wrong it will not yield a larger numerator and hence the loss will be large. When the Dice loss is used during training, after the CEL, it increases the mIoU on the test set by 2% for the networks predictions. This effect is similar for the *DCF-Net 1* and the *DCF-Net 2*.

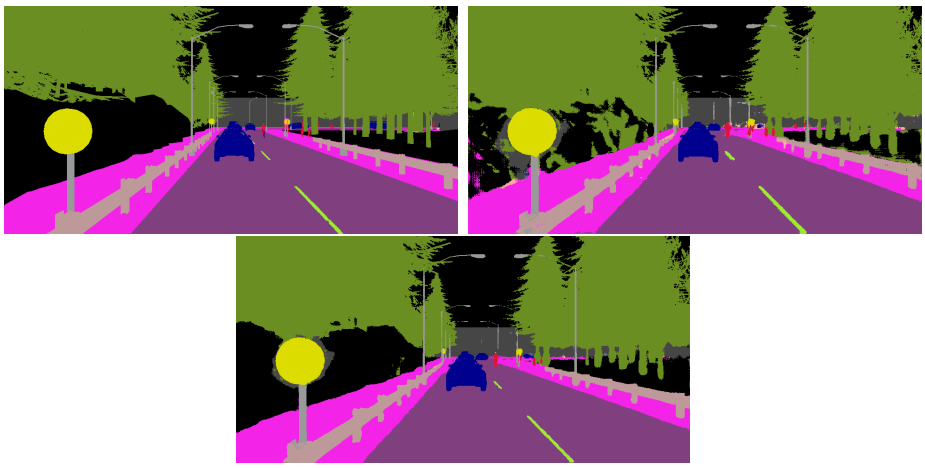


Figure 5.1: Ground truth (top left), result after trainin session 4 using CEL (90.9% pAcc, 70.3% mIoU) (top right), result after training session 5 using Dice loss (95.7% pAcc, 75.9% mIoU) after session five (bottom).

Like mentioned in Section 3.3.1, training using only Dice loss from the beginning does not converge to the optimum. This is because the network parameters need to reach certain area before the Dice loss can be used to refine them. At the beginning of training the parameters in the network are initialized with random values. Hence the initial results are very poor. Since the gradient far away from the global optimum are rather small, the optimizer is likely to get stuck in a local minimum far away

from the global optimum. As mentioned earlier the prediction $\hat{y}$ is multiplied with the ground truth y in the numerator, thus the gradients are very small if the current prediction is far of the ground truth. Hence the network does not improve a lot. Therefore it has turned out to be a good practise to train the network first using CEL with beneficial gradient properties, such that the network reaches a fairly good accuracy and then use the Dice loss to fine tune it in the end to yield the slight improvements shown in Figure 5.1.

5.2 Evaluation of Runtime

The runtimes between the networks differ a few milliseconds. The difference between the *DC-Net* and the *DCF-Net 1* can be explained by the additional computations necessary in the fusion layers. The difference between the *DCF-Net 1* and the *DCF-Net 2* is likely explained by the more efficient WASP layer. We expect this is due to the more efficient cascade structure that is used in the WASP layer, which is reusing the output of the different dilated convolutions for the next dilated convolution. Thus besides the more efficient context information extraction the WASP layer is preferable due to faster runtime.

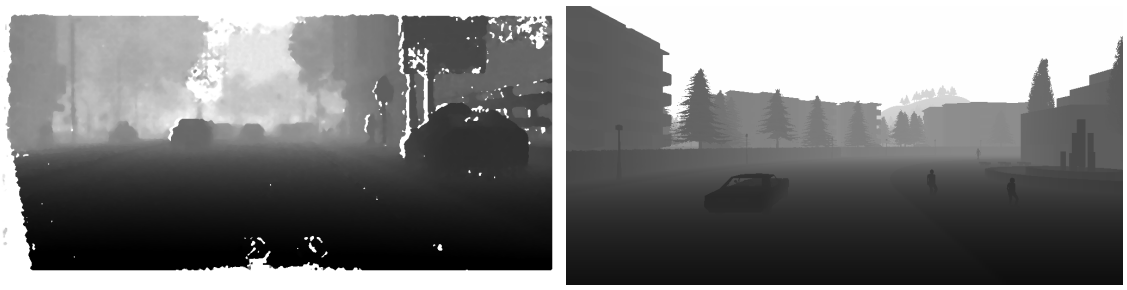
5.3 Evaluation of Resolution

As shown in Table 4.4, using a higher resolution image yields a higher accuracy compared to a lower resolution image. This is mainly due to that more pixels result in more and finer information of structures. A pole for example would in the higher resolution image have about four times as many pixels compared to the low resolution image. It should also be noted that the difference between the mIoUs is greater than the difference between pixel accuracy, 6.7% compared to 1.2% shown in Table 4.4. The difference in pixel accuracy is a lot smaller since incorrect predictions of the small classes does not affect the pixel accuracy a lot. The difference in the IoUs indicates that a network using lower resolution images has a lower accuracy on smaller objects that affect the mIoU as much as the larger classes such as road. This can be seen as well in Figure 4.9, where the difference in the IoU for *building* and *road* is only 1% and 1.2%, whereas for thin structured classes like *pole*, *road line* and *traffic sign* the difference is 19%, 12% and 10% respectively. It should be noted that the accuracy for *building* is comparably high with over 90%. Hence there is only a small room for improvements since the predictions are almost at 100%, compared to class *pole* for example with only 45% for low resolution and 63% for high resolution images.

5.4 Evaluation of the Relevance of Artificial Data

To evaluate the relevance of artifical data the *DCF-Net 2* trained only on the artifical data from Carla is tested on Cityscapes. The result of 30.5% pixel accuracy and 11.8% mIoU, seen in Table 4.5, is low compared to the performance on the Carla

dataset shown in Table 4.3. This is because the objects in the real-world images have a different color and texture. Hence the network tends to confuse a lot of classes. What the network however does is finding the edges of structures such as cars and buildings. Furthermore the depth images of Cityscape have a different structure and quality compared to the Carla depth images. In Carla all objects independent of their distance to the camera have a sharp border, the depth in Cityscapes is not as clearly separated. For example on one side of close objects a white border is produced, as seen in Figure 5.2 (a) on the car parked on the right side. This is not correct since the white color indicates that something is very far away. The reason for these white marks on one side of close objects is that the depth image is calculated from a stereo camera setup and some parts behind close objects are only visible in one image. Thus the disparity of objects behind the close objects can not be calculated and is set to zero, which results in a large depth value after equation (2.5). The reason there is a white edge on the left side of the depth image is that this part is not overlapping with the other image of the stereo camera. In Figure 4.10 it can be seen that the network handles road lines in the Cityscapes dataset, it has not seen before, well. It is due to the high contrast between the road and the road lines. It also helps that the road lines are only distinguished from the RGB image and not from the depth image since the depth image quality is not as good as in Carla.



(a) **(b)**

Figure 5.2: Comparison of depth images from Cityscapes (a) and Carla (b).

The effect of the annotation differences between the Carla and Cityscapes dataset is seen in Figure 4.10. The windows on the building on the left is classified as *vegetation* and not as *building* likely because the tree to the right of the building is reflected in the windows.

5.4.1 Training on Cityscapes

Since the results of the training using only real-world data is better than the results using transfer-learning seen in Table 4.6, it can be concluded that using a network that is pre-trained on artificial data is not giving a better but rather worse starting point for the training than using a network with random initialized parameters. The reason for this is likely that the network trained using transfer-learning gets stuck in a local minima that is close to the global minima for the Carla dataset. But the network only trained on Cityscapes does not get stuck in this minima due to

its different starting point and instead probably finds the global minima and thus optimal parameters. Moreover the total amount of training epochs using transfer-learning is with 190 epochs, almost twice as much as the necessary 110 epochs using the procedure shown in Table 4.1 for real-world data only.

The result of the *DCF-Net 2* on the Cityscape dataset is with 93.0% pixel accuracy and 68.4% mIoU lower than the performance of the network on the Carla dataset (96.9%pAcc, 71.8%mIoU). This slightly lower accuracy is probably due to the quality of the depth images. The Carla depth images have a higher accuracy than the Cityscape images and the edges of object in the Carla depth images are a lot clearer than in the Cityscapes dataset, as shown in Figure 5.2.

5.5 Evaluation of the Relevance of Depth Data

The way adding Gaussian blur to simulate disturbances to the images affects the accuracy for the Carla dataset is seen in Table 4.7. It should be noted that the *DCF-Net 2* performs better than the *RGBC-Net* when the images are not blurred. The drop in mIoU for the *RGBC-Net* is however much more significant than for the *DCF-Net 2* when only the RGB image is blurred. The mIoU for the *RGBC-Net* drops with $48,0\%$ and the corresponding relative drop for the *DCF-Net 2* is 14.6%. This large difference in mIoU drop is mainly due to that the *DCF-Net 2* has a high resolution depth image to rely on for its segmentation, while the *RGBC-Net* only has the RGB image with added blur. This scenario where one has a poor quality RGB image and a high quality depth image is likely to occur if a high resolution LiDAR is used to generate the depth image. Furthermore the poor RGB image is then likely due to fog or dust on the camera lens, since this does not affect the LiDARs depth image.

A weather condition that greatly reduces the vehicles ability to perceive its environment is dense fog. First of all the fog blocks the view for the cameras. But the lasers from the LiDARs are also reflected on the water drops in the air, which decreases the quality of the depth image [44]. To resemble this operating condition when the quality of both the RGB and depth images is decreased both RGB and depth images are blurred. As seen in Table 4.7 the mIoU for the *DCF-Net 2* is now decreased to 37.2% which is still better than the *RGBC-Net*s mIoU of 31.4%. But a lot worse compared to when the quality of the depth images was not blurred. This shows that the depth image provides useful information for the segmentation even if the depth image is blurred.

In Figure 4.13 it can be seen which classes are mostly dependent on the RGB image and which are dependent on the depth image. Thin structures that are very accentuated on the depth image are not affected by the blurred RGB image. Such classes are *pole*, *pedestrian* and *fence*. In these classes the IoU for the *DCF-Net 2* with blurred RGB images is almost as high as the *DCF-Net 2* with no blurred images. A class that depends more on the RGB image is *road line*, here the *DCF-Net 2* with blurred depth images is almost as high as the *DCF-Net 2* with no blurred

images.

If the input of an RGB image is set to zero, i.e. all black, the *DCF-Net 2* is still able to classify traffic signs and poles, since signs and poles are in the depth image in an accentuated manner, whereas the street can not be distinguished from sidewalk and road lines. Buildings and vegetation based on the depth image are also not separable. This information to predict sidewalk, road lines and buildings is obtained in the RGB image. The additional information that the depth map provides is also seen in Figure 4.5 and 4.6. In the segmented ground truth there are thin poles and road lines far away in the image. The network is able to predict the remote poles with higher accuracy compared to the road lines far away. This is due to that the poles are very accentuated in the depth map while the road lines are invisible since they are level with the road.

For the Cityscapes dataset, when the RGB image is blurred the mIoU drops more both absolutely and relatively compared to the Carla dataset. It drops almost as much for the *DCF-Net 2* (CS) as for the *RGBC-Net*. This is not the case for the *DCF-Net 2* (TL) where the drop is only 15.8%. On the other hand, when the depth image is blurred the *DCF-Net 2* (CS) is performing better than the *DCF-Net 2* (TL). When both images are blurred the *DCF-Net 2* (TL) outperforms both other networks. This leads to the assumption that the transfer-learning network, even though it has a lower mIoU on the Cityscapes dataset, is more robust against disturbances. Because the network that the transfer-learning is done from is trained on the Carla dataset which has high quality depth images, it puts more emphasis on the depth images and the network trained with transfer-learning becomes more robust against distubances.

The network only trained on Cityscapes data is on the other hand almost only relying on the RGB data, this is likely due to the poor quality of the depth images in the Cityscapes dataset. This is also supported by the mIoU drop when blurring only the depth image, wich is 1.3 percentage points fot the *DCF-Net 2* (CS). Hence it can be concluded, that the parameters of the *DCF-Net 2* (TL) are in a local minima, which is close to the optimum for the Carla dataset trained *DCF-Net 2* and it is therefore more reliant on the depth images. Whereas the *DCF-Net 2* (CS) is not as reliant on the depth images and therefore in the optimum for the segmentation task without disturbances, which is not making the network reliant on the depth images, due to its poor quality.

Since the *RGBC-Net* is an adaption of the *DCF-Net 2* to only use RGB data it is not an optimal network to perform RGB image segmentation. Nevertheless the results give a reasonable comparison of the additional robustness gained from using depth images along with RGB images.

These assumptions nevertheless rely only on only one robustness study. To ensure the assumptions holds future works could include more tests using other types of disturbances such as noise due to bad light conditions and others.

5.6 Comparison of Results to Recent Works

The most common dataset to rank current networks for traffic environment semantic segmentation is the Cityscapes dataset [37]. On semantic segmentation the current leader is the *HRNet + OCR* from [45] with an mIoU of 84.5%[46]. Networks like these are designed to reach a high accuracy which usually comes with a high computational cost and thus low performance. On the real-time semantic segmentation leaderboard the networks are designed for high performance as well. The highest current mIoU is 75.9% of the *U-HarDNet-70* [47] with a frame rate of 53 fps on a Nvidia GForce RTX 1080 Ti. The *BiSeNet* [7] (see Section 2.1.5) is producing with 68.4% a slightly higher accuracy and with 105 fps a significant faster network than our *DCF-Net 2*. But a difference between them and our network is that they are using only RGB data and no depth data. An overview of the mIoU over runtime for different networks can be seen in Figure 5.3.

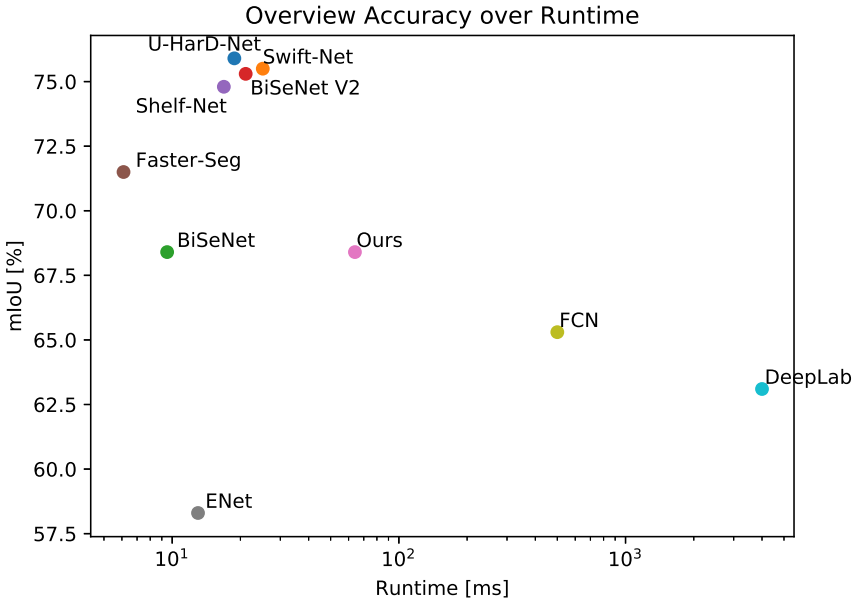


Figure 5.3: Accuracy over runtime for different network approaches.

The accuracy of the *DCF-Net 2* is lower than the leading state-of-the-art networks. Nevertheless the usage of depth images makes our network more robust, than the *RGBC-Net*, when there are disturbances in form of blur in RGB data due to redundancy of information in RGB and depth data. As shown in Section 5.5, our network has a high accuracy on poles and traffic lights even when the RGB image is blurred. The current way to generate the depth images for the Cityscapes dataset uses stereo cameras. This approach does however not yield very good results as discussed earlier. When high resolution LiDARs become more common the quality of the depth images for the real-world will increase a lot [48]. The depth images produced by high resolution LiDARs of the future might have a quality similar to that of the Carla dataset. And at this stage our method comes to its full potential,

since the use of depth images provides complementary and redundant information which makes the segmentation more robust against disturbances.

6

Conclusion

In this thesis a CNN is designed to perform semantic segmentation in traffic environments using RGB-D data. With a framerate of 15.4 fps on a Nvidia GeForce 2080 Ti for the *DCF-Net 2* the required runtime performance of 10 fps was achieved. The designed CNN was developed based on existing networks and extended with a fusion module to increase the effectiveness of the fusion of depth and RGB features. A slight improvement was achieved using this structure with the new fusion module. Furthermore, different context information extraction layers have been tested. It was shown that the *DCF-Net 2* using fusion modules and a WASP layer for extraction of context information performed best. The test was done using artifical data from the simulation environment Carla.

The *DCF-Net 2* was used to evaluate the effect of different image resolutions on the segmentation result. It turned out that the usage of high resolution images is crucial when performing pixel wise semantic segmentation, since a higher image resolution yields a higher accuracy especially for thin structures.

To test the relevance of simulation data the network trained on the Carla dataset was evaluated on real-world data from the Cityscapes dataset. The evaluation was done without training on the Cityscapes images. The accuracy on real-world data was low, however it was shown that the network is able to perceive structures and boundaries of objects. Hence the use of simulation data can not replace the training on real-world data. To evaluate the possibility to use transfer-learning the network trained on Carla data was fine-tuned using the Cityscapes data. This approach yielded good results as the network adapts to the colors and textures and especially the depth images of the real-world data. Nevertheless, training the network only on real-world data is superior to transfer-learning in terms of accuracy.

The use of depth data to increase the robustness of the segmentation was evaluated by adapting the network to use only RGB data. The depth data was shown to improve robustness by a large margin for the Carla dataset and for the Cityscapes dataset when using transfer-learning. The full potential of this approach can however only be obtained when long range high resolution depth images are available.

6.1 Future Works

As previously mentioned in the discussion a robustness study containing more types of disturbances is necessary to increase the understanding of the complementary and redundant nature of RGB and depth images. Furthermore, another approach to train the networks, where the RGB and depth encoders are trained separately is possible. Data augmentation using disturbances already during training, could also yield better results.

The chosen base network was relatively simple, to test the effects of different context extraction layers and our fusion module. As seen in Figure 5.3 there are faster networks for RGB segmentation. Thus the efficient WASP layer and our novel fusion module could be tested with these more complex architectures, to create even faster and more accurate RGB-D segmentation networks.

Bibliography

[1] DARPA. (2008). The grand challenge, [Online]. Available: https://www.darpa.mil/about-us/timeline/-grand-challenge-for-autonomous-vehicles (visited on 02/08/2020).

[2] VOLVO. (2019). Autonomous trucks in real operation, [Online]. Available: https://www.volvotrucks.com/en-en/news/volvo-trucks-magazine/2019/feb/bronnoy.html (visited on 02/08/2020).

[3] T. Inc. (2020). Framtidens körning, [Online]. Available: https://www.tesla.com/sv_SE/autopilot?redirect=no (visited on 03/26/2020).

[4] Waymo. (2020). We're building the worlds most experienced driver, [Online]. Available: https://waymo.com/ (visited on 03/26/2020).

[5] Scania. (2020). Nobina and scania pioneer full length autonomous buses in sweden, [Online]. Available: https://www.nobina.com/en/press/archive/nobina-and-scania-pioneer-full-length-autonomous-buses-in-sweden/# (visited on 03/26/2020).

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", Jun. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[7] C. Yu, J. Wang, C. Peng, C. Gao, G. Yu, and N. Sang, *Bisenet: Bilateral segmentation network for real-time semantic segmentation*, 2018. arXiv: 1808.00897 [cs.CV].

[8] X. Qi, R. Liao, J. Jia, S. Fidler, and R. Urtasun, "3d graph neural networks for rgbd semantic segmentation", *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 5209–5218, 2017.

[9] F. Gu, H. Zhang, and C. Wang, "A classification method for polsar images using slic superpixel segmentation and deep convolution neural network", *IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium*, pp. 6671–6674, 2018.

[10] S.-W. Hung and S.-Y. Lo, "Incorporating luminance, depth and color information by fusion-based networks for semantic segmentation", Sep. 2018.

[11] T. M. Yunlu Chen and E. Gavves, "3d neighborhood convolution: Learning depth-aware features for rgb-d and rgb semantic segmentation", *ArXiv e-prints*, Oct. 2019.

[12] W. Wang and U. Neumann, *Depth-aware cnn for rgb-d segmentation*, 2018. arXiv: 1803.06791 [cs.CV].

[13] S. Lee, S.-J. Park, and K.-S. Hong, "Rdfnet: Rgb-d multi-level residual feature fusion for indoor semantic segmentation", Oct. 2017, pp. 4990–4999. DOI: 10.1109/ICCV.2017.533.

[14] A. Valada, R. Mohan, and W. Burgard, "Self-supervised model adaptation for multimodal semantic segmentation", *International Journal of Computer Vision (IJCV)*, Jul. 2019, Special Issue: Deep Learning for Robotic Vision, ISSN: 1573-1405. DOI: 10.1007/s11263-019-01188-y.

[15] M. S. Nixon and A. S. Aguado, "Chapter 4 - low-level feature extraction (including edge detection)", in *Feature Extraction and Image Processing for Computer Vision (Third Edition)*, M. S. Nixon and A. S. Aguado, Eds., Third Edition, Oxford: Academic Press, 2012, pp. 137–216, ISBN: 978-0-12-396549-3. DOI: https://doi.org/10.1016/B978-0-12-396549-3.00004-5. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780123965493000045.

[16] J. Shotton, J. Winn, C. Rother, and A. Criminisi, "Textonboost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context", *Int. Journal of Computer Vision (IJCV)*, Jan. 2009. [Online]. Available: https://www.microsoft.com/en-us/research/publication/textonboost-for-image-understanding-multi-class-object-recognition-and-segmentation-by-jointly-modeling-texture-layout-and-context/.

[17] K. O'Shea and R. Nash, "An introduction to convolutional neural networks", *ArXiv e-prints*, Nov. 2015.

[18] Y. Liu, Y. Wang, S. Wang, T. Liang, Q. Zhao, Z. Tang, and H. Ling, *Cbnet: A novel composite backbone network architecture for object detection*, 2019. arXiv: 1909.03625 [cs.CV].

[19] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation", May 2015.

[20] L. 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", *CoRR*, vol. abs/1606.00915, 2016. arXiv: 1606.00915. [Online]. Available: http://arxiv.org/abs/1606.00915.

[21] B. Artacho and A. E. Savakis, "Waterfall atrous spatial pooling architecture for efficient semantic segmentation", *Sensors (Basel, Switzerland)*, vol. 19, 2019.

[22] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation", Jun. 2017.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", *CoRR*, vol. abs/1512.03385, 2015. arXiv: 1512.03385. [Online]. Available: http://arxiv.org/abs/1512.03385.

[24] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient transfer learning", *CoRR*, vol. abs/1611.06440, 2016. arXiv: 1611.06440. [Online]. Available: http://arxiv.org/abs/1611.06440.

[25] T. Gröndahl and A. Samuelsson, "Self-Supervised Cross-Connected CNNs for Binocular Disparity Estimation", Master's thesis, Chalmers University of Technology, 2018.

[26] C. Godard, O. Mac Aodha, M. Firman, and G. J. Brostow, "Digging into self-supervised monocular depth prediction", Oct. 2019.

[27] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator", in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[28] T. Wang, H. Hu, C. H. Lin, Y. Tsai, W. Chiu, and M. Sun, "3d lidar and stereo fusion using stereo matching network with conditional cost volume normalization", *CoRR*, vol. abs/1904.02917, 2019. arXiv: 1904.02917. [Online]. Available: http://arxiv.org/abs/1904.02917.

[29] D. Lin, G. Chen, D. Cohen-Or, P. Heng, and H. Huang, "Cascaded feature network for semantic segmentation of rgb-d images", in *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017, pp. 1320–1328. DOI: 10.1109/ICCV.2017.147.

[30] *Nyu depth v2 test leaderboard | papers with code*, https://paperswithcode.com/sota/semantic-segmentation-on-nyu-depth-v2, Accessed: 2020-05-19.

[31] X. Hu, K. Yang, L. Fei, and K. Wang, "Acnet: Attention based network to exploit complementary features for rgbd semantic segmentation", Sep. 2019, pp. 1440–1444. DOI: 10.1109/ICIP.2019.8803025.

[32] J. Jiang, L. Zheng, F. Luo, and Z. Zhang, "Rednet: Residual encoder-decoder network for indoor RGB-D semantic segmentation", *CoRR*, vol. abs/1806.01054, 2018. arXiv: 1806.01054. [Online]. Available: http://arxiv.org/abs/1806.01054.

[33] S. Ruder, "An overview of gradient descent optimization algorithms", *CoRR*, vol. abs/1609.04747, 2016. arXiv: 1609.04747. [Online]. Available: http://arxiv.org/abs/1609.04747.

[34] *Pytorch documentation*, https://pytorch.org/docs/stable/torchvision/models.html, Accessed: 2020-05-19.

[35] F. Milletari, N. Navab, and S. Ahmadi, "V-net: Fully convolutional neural networks for volumetric medical image segmentation", *CoRR*, vol. abs/1606.04797, 2016. arXiv: 1606.04797. [Online]. Available: http://arxiv.org/abs/1606.04797.

[36] P. K. Nathan Silberman Derek Hoiem and R. Fergus, "Indoor segmentation and support inference from rgbd images", in *ECCV*, 2012.

[37] 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.

[38] *Pytorch documentation*, https://pytorch.org/, Accessed: 2020-03-06.

[39] M. Y. Yang, B. Rosenhahn, and V. Murino, *Multimodal Scene Understanding: Algorithms, Applications and Deep Learning*. Aug. 2019, pp. 41–62, ISBN: 9780128173596. DOI: 10.1016/C2018-0-01791-0.

[40] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet

Large Scale Visual Recognition Challenge", *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015. DOI: 10.1007/s11263-015-0816-y.

[41] *Nvidia gpu specifications*, https://www.nvidia.com/en-eu/geforce/graphics-cards/rtx-2080-ti/, Accessed: 2020-03-06.

[42] L. Prechelt, "Early stopping - but when?", Mar. 2000. DOI: 10.1007/3-540-49430-8_3.

[43] *Metrics to evaluate your semantic segmentation model*, https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb99639aa2, Accessed: 2020-06-02.

[44] R. Heinzler, P. Schindler, J. Seekircher, W. Ritter, and W. Stork, "Weather influence and classification with automotive lidar sensors", Jun. 2019, pp. 1527–1534. DOI: 10.1109/IVS.2019.8814205.

[45] Y. Yuan, X. Chen, and J. Wang, *Object-contextual representations for semantic segmentation*, 2019. arXiv: 1909.11065 [cs.CV].

[46] *Cityscapes test leaderboard | papers with code*, https://paperswithcode.com/sota/semantic-segmentation-on-cityscapes, Accessed: 2020-03-12.

[47] P. Chao, C.-Y. Kao, Y.-S. Ruan, C.-H. Huang, and Y.-L. Lin, *Hardnet: A low memory traffic network*, 2019. arXiv: 1909.00948 [cs.CV].

[48] Robosence. (2020). Rs-lidar-m1, [Online]. Available: https://www.robosense.ai/rslidar/rs-lidar-M1 (visited on 03/26/2020).

A

ResNet structure

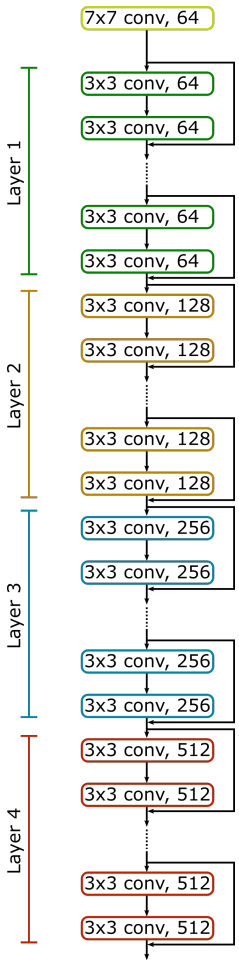


Figure A.1: Structure of the *ResNet-xx* encoder used for the network. ($k \times k$ conv, c), where k is the filter size and c is the number of channels. Each layer has a different amount of convolution layers depending on the version of the *ResNet*.

B

Class Pixel accuracy and IoU

B.1 Networks Trained on the Carla Dataset

Table B.1: Pixel accuracy and IoU for the *DC-Net* trained on the Carla dataset.

Class	**Pixel Accuracy**	**mIoU**
Unlabeled	97.7%	96.7%
Building	96.5%	93.9%
Fence	37.2%	34.6%
Other	44.2%	37.0%
Pedestrian	60.4%	49.2%
Pole	66.5%	59.9%
Road line	69.5%	63.7%
Road	97.8%	95.9%
Sidewalk	96.5%	92.2%
Vegetation	88.2%	77.8%
Car	93.5%	79.9%
Wall	52.9%	43.9%
Traffic Sign	63.0%	52.7%

Table B.2: Pixel accuracy and IoU for the *DCF-Net 1* trained on the Carla dataset.

Class	**Pixel Accuracy**	**mIoU**
Unlabeled	98.4%	96.8%
Building	96.2%	94.1%
Fence	42.0%	36.4%
Other	51.9%	42.0%
Pedestrian	64.8%	52.0%
Pole	68.1%	61.9%
Road line	72.8%	65.3%
Road	97.7%	96.2%
Sidewalk	97.1%	93.0%
Vegetation	92.0%	79.3%
Car	92.8%	81.5%
Wall	54.0%	45.8%
Traffic Sign	65.7%	54.3%

Table B.3: Pixel accuracy and IoU for the *DCF-Net 2* trained on the Carla dataset.

Class	**Pixel Accuracy**	**mIoU**
Unlabeled	98.4%	95.5%
Building	94.6%	93.0%
Fence	46.7%	40.2%
Other	51.6%	43.5%
Pedestrian	61.5%	50.1%
Pole	72.1%	63.3%
Road line	71.0%	64.4%
Road	98.2%	95.9%
Sidewalk	96.3%	92.5%
Vegetation	89.5%	79.4%
Car	94.2%	81.2%
Wall	55.0%	45.8%
Traffic Sign	64.3%	54.4%

Table B.4: Pixel accuracy and IoU for the *DCF-Net 2* trained on the Carla dataset with image resolution (512×256).

Class	**Pixel Accuracy**	**mIoU**
Unlabeled	97.8%	94.9%
Building	95.2%	92.5%
Fence	47.5%	37.6
Other	42.6%	36.9%
Pedestrian	52.4%	41.2%
Pole	48.7%	45.0%
Road line	58.9%	52.1%
Road	97.2%	94.3%
Sidewalk	95.1%	89.6%
Vegetation	78.1%	70.4%
Car	78.4%	66.2%
Wall	49.7%	40.7%
Traffic Sign	51.6%	41.9%

B.2 Networks Trained on the Cityscapes Dataset

Table B.5: Pixel accuracy and IoU for the *DCF-Net 2* pre-trained on the Carla dataset, transfer-learning to the Cityscapes dataset

Class	Pixel Accuracy	mIoU
Unlabeled	87.6%	77.5%
Building	85.6%	78.7%
Fence	36.3%	26.2%
Pedestrian	64.7%	48.7%
Pole	49.3%	35.5%
Road	89.1%	86.8%
Sidewalk	70.6%	61.8%
Vegetation	90.2%	82.1%
Car	84.7%	52.8%
Traffic Sign	56.3%	43.4%

Table B.6: Pixel accuracy and IoU for the *DCF-Net 2* trained only on Cityscapes data.

Class	Pixel Accuracy	mIoU
Unlabeled	79.7%	75.5%
Building	90.8%	82.6%
Fence	56.2%	46.0%
Pedestrian	72.3%	56.7%
Pole	49.7%	38.0%
Road	97.0%	93.7%
Sidewalk	82.6%	73.7%
Vegetation	94.3%	87.5%
Car	90.2%	79.8%
Traffic Sign	65.9%	51.9%

B.3 Robustness Study

B.3.1 Carla Dataset

Table B.7: Pixel accuracy and IoU for the *DCF-Net 2* without blur.

Class	Pixel Accuracy	mIoU
Unlabeled	98.4%	95.5%
Building	94.6%	93.0%
Fence	46.7%	40.2%
Other	51.6%	43.5%
Pedestrian	61.5%	50.1%
Pole	72.1%	63.3%
Road line	71.0%	64.4%
Road	98.2%	95.9%
Sidewalk	96.3%	92.5%
Vegetation	89.5%	79.4%
Car	94.2%	81.2%
Wall	55.0%	45.8%
Traffic Sign	64.3%	54.4%

Table B.8: Pixel accuracy and IoU for the *RGBC-Net* without blur.

Class	Pixel Accuracy	mIoU
Unlabeled	91.5%	88.4%
Building	87.7%	85.6%
Fence	47.4%	33.5%
Other	41.7%	27.3%
Pedestrian	46.9%	34.0%
Pole	44.0%	36.8%
Road line	66.6%	61.5%
Road	97.7%	94.3%
Sidewalk	94.2%	87.1%
Vegetation	85.2%	74.4%
Car	88.4%	54.0%
Wall	50.1%	30.6%
Traffic Sign	31.6%	26.4%

Table B.9: Pixel accuracy and IoU for the *DCF-Net 2* with blurred RGB data.

Class	**Pixel Accuracy**	**mIoU**
Unlabeled	98.3%	93.4%
Building	91.5%	90.0%
Fence	42.3%	35.0%
Other	38.5%	32.5%
Pedestrian	54.0%	44.7%
Pole	67.0%	55.9%
Road line	7.1%	6.8%
Road	96.2%	90.8%
Sidewalk	95.3%	85.3%
Vegetation	81.6%	73.6%
Car	84.8%	78.9%
Wall	55.9%	40.4%
Traffic Sign	49.9%	44.4%

Table B.10: Pixel accuracy and IoU for the *DCF-Net 2* with blurred depth data.

Class	**Pixel Accuracy**	**mIoU**
Unlabeled	95.9%	74.6%
Building	84.7%	75.0%
Fence	9.2%	8.5%
Other	23.8%	16.9%
Pedestrian	5.0%	4.5%
Pole	17.5%	14.7%
Road line	69.0%	53.1%
Road	97.8%	93.4%
Sidewalk	92.8%	84.0%
Vegetation	41.0%	40.1%
Car	67.4%	57.8%
Wall	21.6%	18.4%
Traffic Sign	8.9%	8.7%

Table B.11: Pixel accuracy and IoU for the *DCF-Net 2* with blurred RGB and depth data.

Class	Pixel Accuracy	mIoU
Unlabeled	96.4%	71.5%
Building	78.4%	72.8%
Fence	4.7%	4.4%
Other	18.6%	12.5%
Pedestrian	3.9%	3.4%
Pole	14.6%	12.1%
Road line	6.3%	5.8%
Road	95.1%	88.0%
Sidewalk	91.6%	74.8%
Vegetation	33.0%	32.2%
Car	51.6%	47.6%
Wall	29.4%	19.8%
Traffic Sign	3.8%	3.7%

Table B.12: Pixel accuracy and IoU for the *RGBC-Net* with blurred RGB data.

Class	Pixel Accuracy	mIoU
Unlabeled	84.3%	72.7%
Building	54.5%	51.8%
Fence	11.8%	8.8%
Other	22.1%	6.1%
Pedestrian	10.5%	7.0%
Pole	17.0%	12.0%
Road line	14.1%	12.0%
Road	94.0%	78.1%
Sidewalk	66.2%	48.5%
Vegetation	36.7%	33.9%
Car	60.0%	25.5%
Wall	29.6%	7.5%
Traffic Sign	3.3%	3.2%

B.3.2 Cityscapes Dataset

Table B.13: Pixel accuracy and IoU for the *DCF-Net 2* (TL) without blur.

Class	Pixel Accuracy	mIoU
Unlabeled	87.6%	77.6%
Building	85.7%	78.9%
Fence	36.8%	26.5%
Pedestrian	64.6%	48.8%
Pole	48.9%	35.1%
Road	89.2%	86.9%
Sidewalk	70.9%	62.1%
Vegetation	90.1%	81.9%
Car	84.9%	53.2%
Traffic Sign	56.7%	43.6%

Table B.14: Pixel accuracy and IoU for the *DCF-Net 2* (CS) without blur.

Class	Pixel Accuracy	mIoU
Unlabeled	87.6%	77.5%
Building	85.6%	78.7%
Fence	36.3%	26.2%
Pedestrian	64.7%	48.7%
Pole	49.3%	35.5%
Road	89.1%	86.8%
Sidewalk	70.6%	61.8%
Vegetation	90.2%	82.1%
Car	84.7%	52.8%
Traffic Sign	56.3%	43.4%

Table B.15: Pixel accuracy and IoU for the *RGBC-Net* without blur.

Class	Pixel Accuracy	mIoU
Unlabeled	82.3%	75.5%
Building	82.9%	75.5%
Fence	49.5%	32.9%
Pedestrian	66.1%	44.5%
Pole	37.5%	29.2%
Road	95.5%	89.6%
Sidewalk	54.7%	49.0%
Vegetation	89.3%	82.6%
Car	84.7%	53.3%
Traffic Sign	53.4%	40.5%

Table B.16: Pixel accuracy and IoU for the *DCF-Net 2* (TL) with blurred RGB data.

Class	Pixel Accuracy	mIoU
Unlabeled	83.3%	67.8%
Building	89.0%	66.0%
Fence	4.8%	4.3%
Pedestrian	50.3%	36.4%
Pole	22.1%	17.5%
Road	88.8%	85.2%
Sidewalk	60.1%	50.6%
Vegetation	61.5%	53.3%
Car	64.0%	40.7%
Traffic Sign	12.0%	11.1%

Table B.17: Pixel accuracy and IoU for the *DCF-Net 2* (CS) with blurred RGB data.

Class	Pixel Accuracy	mIoU
Unlabeled	70.1%	50.7%
Building	95.1%	35.8%
Fence	0.3%	0.3%
Pedestrian	26.1%	18.2%
Pole	11.6%	8.9%
Road	29.5%	28.6%
Sidewalk	30.2%	15.5%
Vegetation	31.3%	28.7%
Car	36.6%	31.9%
Traffic Sign	6.8%	6.5%

Table B.18: Pixel accuracy and IoU for the *DCF-Net 2* (TL) with blurred depth data.

Class	Pixel Accuracy	mIoU
Unlabeled	84.9%	64.7%
Building	81.8%	72.8%
Fence	22.0%	16.3%
Pedestrian	53.8%	40.6%
Pole	34.0%	24.5%
Road	70.4%	68.7%
Sidewalk	62.2%	54.2%
Vegetation	89.5%	78.4%
Car	82.6%	33.7%
Traffic Sign	38.7%	31.0%

Table B.19: Pixel accuracy and IoU for the *DCF-Net 2* (CS) with blurred depth data.

Class	Pixel Accuracy	mIoU
Unlabeled	75.1%	70.3%
Building	91.6%	80.7%
Fence	48.8%	40.8%
Pedestrian	69.1%	54.8%
Pole	46.4%	35.6%
Road	96.5%	91.8%
Sidewalk	83.0%	72.9%
Vegetation	92.1%	86.5%
Car	89.4%	79.3%
Traffic Sign	58.9%	49.1%

Table B.20: Pixel accuracy and IoU for the *DCF-Net 2* (TL) with blurred RGB and depth data.

Class	Pixel Accuracy	mIoU
Unlabeled	80.6%	55.9%
Building	85.7%	62.5%
Fence	1.3%	1.3%
Pedestrian	40.2%	27.9%
Pole	13.2%	10.1%
Road	73.4%	70.7%
Sidewalk	48.7%	40.4%
Vegetation	64.2%	51.6%
Car	55.0%	26.5%
Traffic Sign	6.3%	6.0%

Table B.21: Pixel accuracy and IoU for the *DCF-Net 2* (CS) with blurred RGB and depth data.

Class	Pixel Accuracy	mIoU
Unlabeled	58.8%	39.3%
Building	96.4%	32.8%
Fence	0.1%	0.1%
Pedestrian	15.7%	12.0%
Pole	7.1%	5.5%
Road	26.0%	25.0%
Sidewalk	28.8%	15.1%
Vegetation	17.7%	16.6%
Car	24.6%	21.3%
Traffic Sign	3.0%	2.9%

Table B.22: Pixel accuracy and IoU for the *RGBC-Net* with blurred RGB data.

Class	**Pixel Accuracy**	**mIoU**
Unlabeled	57.3%	34.4%
Building	46.4%	34.5%
Fence	1.1%	0.9%
Pedestrian	42.3%	9.0%
Pole	15.1%	9.7%
Road	49.5%	46.8%
Sidewalk	26.1%	20.5%
Vegetation	23.0%	22.4%
Car	66.3%	14.6%
Traffic Sign	7.6%	6.2%

DEPARTMENT OF MECHANICS AND
MARITIME SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY