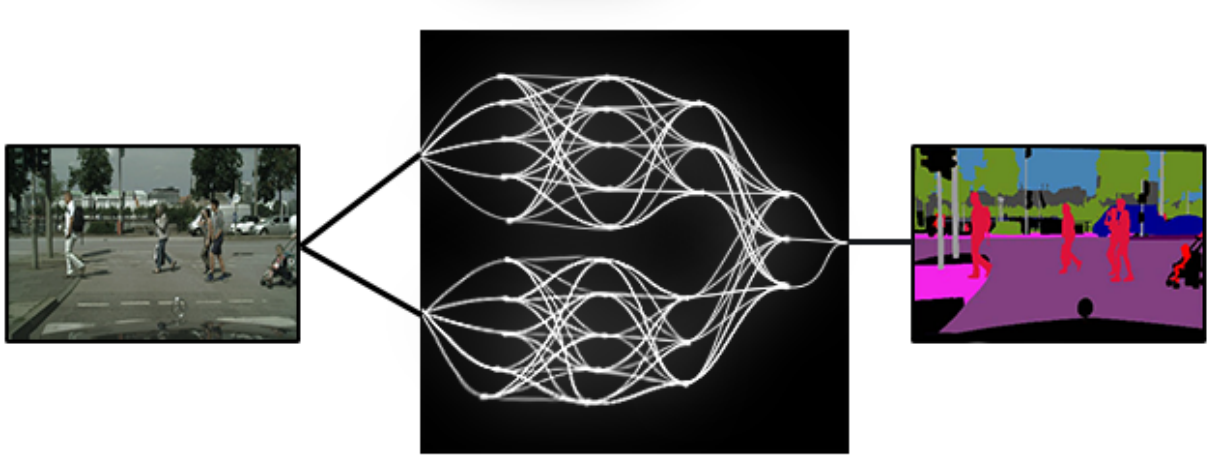
CHALMERS
UNIVERSITY OF TECHNOLOGY



# Deep Neural Network Fusion for Situational Awareness

## Fusion of semantic segmentation and unsupervised depth estimation models

Master's thesis in System, Control and Mechatronics

ANNIE EDVARDSSON
MARTIN TRIEU

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

# Deep Neural Network Fusion for Situational Awareness

Fusion of semantic segmentation and unsupervised depth estimation models

ANNIE EDVARDSSON

MARTIN TRIEU

Supervisor: JENS HENRIKSSON

Examiner: Irene Yu-Hua Gu


CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
*Division of Systems, Control and Mechatronics*
Chalmers University of Technology
Gothenburg, Sweden 2019

Deep Neural Network Fusion for Situational Awareness
Fusion of semantic segmentation and unsupervised depth estimation models
ANNIE EDVARDSSON
MARTIN TRIEU

© ANNIE EDVARDSSON, 2019.
© MARTIN TRIEU, 2019.

Supervisor: Jens Henriksson, Semcon Sweden AB
Examiner: Irene Yu-Hua Gu, Electrical Engineering, Chalmers

Department of Electrical Engineering
Division of Systems, Control and Mechatronics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A illustration of a fusion between two deep neural networks receiving an RGB-image as input and output a pixel-wise classification. Input and output image from Cityscapes dataset [15].

Typeset in LaTeX
Gothenburg, Sweden 2019

Deep Neural Network Fusion for Situational Awareness
Fusion of semantic segmentation and unsupervised depth estimation models
ANNIE EDVARDSSON
MARTIN TRIEU
Department of Electrical Engineering
Chalmers University of Technology

# Abstract

The Computer vision field has evolved drastically since 2012 when AlexNet won ImageNet by utilizing Convolutional Neural Network (CNN). With further advancement, industries started to take interest and are now using CV for different implementations.

In the automotive industry CV take part in the development of complete situational awareness around a vehicle. The CV problems semantic segmentation and depth estimation are crucial for scene understanding and researchers have proposed several methods to solve these.

This thesis studies if a fusion of two deep neural networks can improve their performances for their respective tasks. The fusion is done by generating depth estimation images from a CNN that are then used as input together with a RGB image to solve semantic segmentation. Five different fusion networks were proposed and compared with a baseline network. The fusion networks are all designed as Convolutional Neural Networks with an autoencoder architecture. Their differences are how the depth image is processed and the complexity of their architectures.

Several results was generated to evaluate the performances of the networks. Mean Intersection over Union (MIoU) was the metric used to compare the accuracy and the confusion matrix were investigated for a more detailed comparison. The generalization of the networks were also compared and their estimated semantic segmentation images as well. The models demonstrate similar performance and shows that a fusion of two deep neural networks neither decreased nor increased their performances with the proposed method. With this said another method to approach this study could possibly yield different results since the depth image is generated from a CNN the information that can be extracted from an encoder is similar to those in the RGB image. Therefore, it cannot be confirmed if a fusion between two deep neural networks can affect their performances.

Keywords: CNN, fusion, CV, deep neural network, semantic segmentation, unsupervised, depth estimation, robustness.

# Acknowledgements

We would like to thank our supervisor Jens Henriksson that continuously contributed with discussions and guidance throughout the thesis. He has given us a lot of advice within the field of computer vision and done a lot of proofreading.

We would also like to thank Irene Yu-Hu Gua who contributed as our examiner and gave us a valuable intuition of working in the field of research.

Lastly, we would like to thank Semcon Sweden AB who provided us with equipment to complete this project and good colleagues who kept us cheered up during the whole period. Especially our fellow thesis companions Olle and Daniel for their invaluable opinion regarding matter, significance and style of the input data for Figure 2.2.

Annie Edvardsson, Gothenburg, June 2019
Martin Trieu, Gothenburg, June 2019

# Contents

# List of Figures

# List of Tables

# 1

# Introduction

Within computer vision (CV) there exists a variation of different tasks that can be achieved with deep learning. One impressive task that have evolved from image-level classification is pixel-wise classification on an image [39, 5, 50, 12], there are several industries that these networks can be applied to and one common is autonomous driving. Additional examples of industries include agriculture [6], facial segmentation [68], Geosensing [48] and medical image diagnostic [82].

In the past, depth data have mostly been acquired by sensors such as LiDAR [34], but in 2016 an unsupervised method to train a monocular depth estimation model was designed [64, 30]. Perceiving depth in a scene is an important task in robotics, as its crucial for obstacle avoidance, navigation and planning. What these problem solving methods have in common is that they use a convolutional neural network (CNN) which have been the most common network within CV since 2012 [42, 71].

## 1.1 Background

The popularity of computer vision (CV) have increased since AlexNet won the annually ImageNet Large Scale Visual Recognition Challenge in 2012 (ILSVRC2012) and proved the utility of a CNN [67, 42], but the scientific field have been around longer than that. When talking about CV the networks mostly consists of convolutional layers and are known as CNNs. The history behind how CV and CNNs has evolved can be traced back to 1959 when two neurophysiologists, David Hubel and Torsten Wiesel, described the core response properties of the neurons in the visual cortex by running some experiments on the brain of a cat [35]. They found out that there are neurons that either activates for simple or complex structures in the visual cortex and that the neurons that activates first during visual processing are the neurons that process simple structures. The same principle appears in CNNs where the features becomes more complex when advancing deeper in a network [42].

In 1980 the first convolutional layer appeared in the neural network called *Neocognitron* which was proposed by the computer scientist Kunihiko Fukushima. The network, similar to the findings by Hubel and Wiesel [35], was built of simple and complex cells which could recognize patterns unaffected by position shifts. The architecture contained several convolutional layers with rectangular receptive fields

and weight parameters [26]. In 1989 Yann LeCun applied backpropagation learning to the *Neocognitron* and later in his research he proposed the pioneering network *LeNet-5*. But, the ability to process images with high resolutions were constrained by computing resources [46].

In 2004 K.-S. Oh and K. Jung proved that neural networks could be greatly accelerated on GPUs [58] and in 2006 the first CNN running on a GPU, which was four times faster than on a CPU, was proposed by K. Chellapilla *et al.*[10]. Soon thereafter D. C. Ciresan *et al.* proved in 2010 that a standard deep neural network could be trained with backpropagation more efficiently on a GPU and outperformed previous methods on the MNIST handwritten digits benchmark [45] with their network [14]. Later they implemented the same GPU method to CNNs in 2011 and won four image competitions between May 15, 2011 and September 30, 2012 [71]. At the same time the ILSVRC2012 winner AlexNet stated that their architecture was quite similar to the one from Dan C. Ciresan *et al.* [42, 14]. With the contributions from Dan C. Cireson *et al.* and the team behind AlexNet, 2012 have been associated as the breakthrough year for deep convolutional neural networks [43, 19]. Since then the top performing models for different challenges in the CV field have been CNNs [67, 15, 2, 81, 23].

After the breakthrough new tasks within CV started to arise. At first, developers used MNIST, PascalVOC and ILVSVRC as datasets to benchmark the model's performances in image classification. Then additional challenges was introduced, such as object detection and localization, scene classification and parsing, and pixel-wise segmentation [67, 23]. With the promising development in CV industries started to take interest in it and have put a lot of resources in the field since then and seems to keep on doing so according to market forecasts [69]. With the newfound interest by industries, public datasets featuring road scenes started to appear, such as KITTI, Cityscapes and BDD100k [2, 15, 81]. These datasets are most relevant for the automotive industry and feature CV tasks such as object detection and localization, semantic segmentation, and depth prediction [5, 64, 2].

## 1.2 Aim

The aim of this project is to study how a fusion of two deep neural networks with different tasks will affect their performances.

## 1.3 Scope

To achieve the aim of this project different networks with various tasks can be used. However, to make this project feasible within the time frame of this project two networks which are designed for the tasks, semantic segmentation and unsupervised depth estimation, are used. The reason why those were chosen was because they used the same input data which make a fusion of those easily applicable.

## 1.4 Limitations

The aim is to study how a fusion affect the networks but the project is limited by time and therefore we only train the semantic segmentation network while keeping the second network pre-trained. The choice to use pre-trained weights also limit the project to which type of dataset that can be used. Since the network with the pre-trained weights is trained on a dataset with road scenes the only dataset that can be used is those that contains road scenes.

The related work about multi-task learning seems to discuss valid methods to implement for this study. But this type of concept was found too late in to the project and unfortunately not a part of the method. Although, it is discussed to investigate the possibilities to fuse the networks through multi-task learning for future work.

## 1.5 Related work

Some researches are closely related to the project scope and aim, these will be presented in this chapter, this includes methods for semantic segmentation, unsupervised depth estimation and multi-task learning.

### 1.5.1 Semantic segmentation

Semantic segmentation is an important problem to overcome to achieve complete situational awareness. Multiple methods have been proposed to solve this task, examples are ResNet [33], SegNet [5] and DeepLabv3+ [12].

The residual network *ResNet* was the winner of ILVSVRC 2015 [67]. It utilize skip connections which are additional connections between nodes in different layers, the connections skips one or more layers during processing [59]. These connections contributes by mitigate the vanishing gradient problem, i.e. the problem that arises when the gradients of the loss function approaches zero, and make it possible to build deeper network without the error saturating or increasing.

The first autoencoder architecture used for semantic segmentation was proposed by V. Badrinarayanan *et al.* and is called SegNet [5]. Autoencoders consists of an encoder and decoder, the decoder is a series of downsampling convolutional layers and the decoder is a mirrored reflection of the encoder with upsampling instead of downsampling. Another characteristic feature of SegNet is the storing of indices during downsampling, these are afterwards used in the upsampling to construct a high resolution feature map.

An architecture that performed very good in the Cityscapes benchmark is *DeepLabv3+*, which uses Atrous Spatial Pyramid Pooling [15]. Atrous Convolution, also called dilated convolution, is a type of convolutional filer with spaces between the values in the filter, which result in a larger range of view without increasing the number

of parameters. DeepLabv3+ uses these Atrous convolutional layers with different rates in a Spatial Pyramid, this helps the network to account for different object scales [11].

### 1.5.2 Unsupervised depth estimation

The paper *Unsupervised Monocular Depth Estimation with Left-Right Consistency* from Godard *et al.* on the Conference on Computer Vision and Pattern Recognition 2017 (CVPR2017) proposed a method for unsupervised depth estimation that other research is now building upon [29, 17]. The paper presents a method that use stereo-pair images during training to learn the disparity function that can map the left image to the right and vice versa. When the disparity model is learned it can be used to calculate the depth of a single image. The advantages with the proposed method from Godard *et al.* is, that it is unsupervised and therefore do not require annotated data, which in many cases are expensive and hard to come by, and only require a single image as an input after training which makes it more applicable.

Unsupervised depth estimation networks have in general been deep and complex and required a high computational cost. Poggi *et al.* confront the problem by presenting a pyramidal architecture, *PyD-Net*, that can run on a CPU [64], which is based of the unsupervised method proposed by Godard *et al.*, described above [29].

### 1.5.3 Multi-task learning

A variety of problems can be solved by feeding a CNN with an input image, for example *semantic segmentation* [5, 33, 11], *image classification* [42, 33, 46], and *unsupervised depth estimation* [30, 64], but the problems are often solved individually. By solving the multiple problems simultaneously with a single model it can increase the efficiency during learning and inference and increase performance in some cases [40, 75], this concept is called Multi-task learning (MTL). Although, a problem with MTL have been the tuning of the multi-loss function [49], but A. Kendall *et al.* proposed a method to solve the problem by considering the homoscedastic uncertainty of each task when weighting the losses [40]. They tested the concept on a model that solved *semantic segmentation*, *monocular depth estimation* and *instance segmentation* with results that showed good performance and sometimes even better than models that only focused on one task [40].

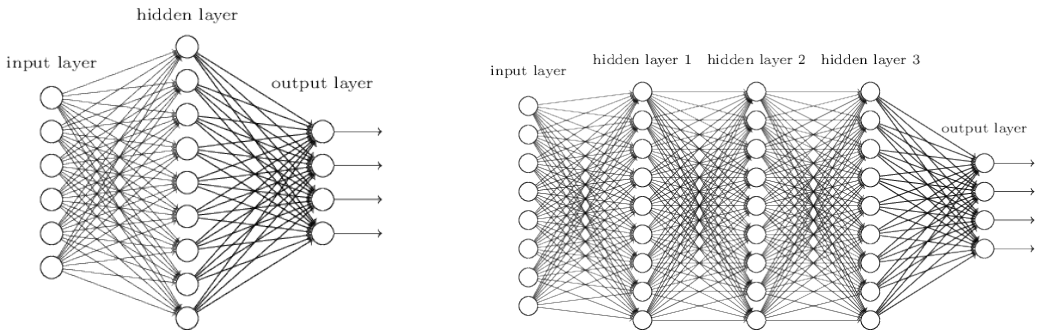
# 2

# Theory

## 2.1 Deep learning

An artificial neural network is a collection of neurons that initially was inspired by the brain of mammals and is used as a tool to find patterns, more known as features. A neural network consists of layers with neurons, there exists three types of layers, the input layer, hidden layers and output layer. The hidden layers are always in between the input and output layers. An example of a neural network with a single hidden layer of neurons can be seen in Figure 2.1a.

Deep learning is a category of machine learning algorithms, the definition of a deep neural network is to have numerous hidden layers, an example of a deep neural network with three hidden layers can be seen in Figure 2.1b. There is no exact definition of how many hidden layers there are in a deep neural network, but usually a network is called deep if the network has two or more hidden layers, whereas a one hidden layer network is called shallow. The advantage of a deep neural network is the ability to learn multiple levels of features. The deeper layers of a network will learn complex shapes, for example, faces or digits while the shallow layers learn more abstract features like edges. This makes it possible for a network to learn how to solve complex classification tasks [62].




**(a)** Shallow neural network **(b)** Deep neural network

**Figure 2.1:** An example of a shallow and a deep neural network. Figures originally from Michael Nielsen's article [56]

## 2.2 Deep Convolutional Neural Networks

Convolutional Neural Network (CNN) is a type of deep neural network which is commonly adapted in computer vision or analysis of other visual images. There is a lot of image processing tasks which utilize CNNs, one example is a classifier which task is to identify a class between a fixed number of classes. A classification task to distinguish if there is a cat in an image or not will be used as an example to describe the different building blocks that appear in a CNN structure, the architecture can be seen in Figure 2.2. It is a task that can be learned through supervised learning, which refers to the network being provided with an input and ground truth output. In this example the input is an image and the output is an array with 2 elements, representing the probability of cat respectively no cat.

Leaky ReLU is one of several solutions to overcome a problem called *dying ReLU problem* which occurs when using ReLU [51]

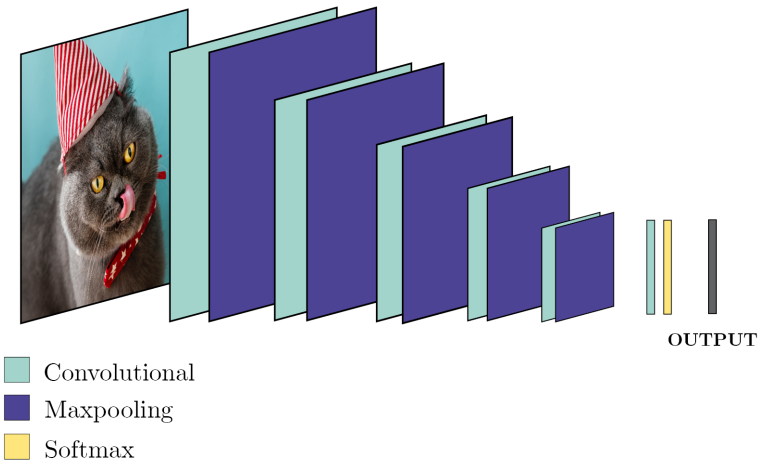



**Figure 2.2:** An example of a simple Convolutional Neural Network to classify if there is a cat in an image

The hidden layers in a CNN consists of convolutional layers, pooling layers, activation functions and regularizers. Since there are a lot of different variations of the mentioned layers and operations the ones that are used in this project will only be discussed.

### 2.2.1 Convolutional layer

The most associated layers when speaking of CNN is convolutional layers. A description of convolutional layers was well put by Adit Deshpande were he briefly explains a CNN [18]. He described a convolutional layer as a flashlight that is shining through the image starting from the top left. Imagine that the area covered by

the shining light is $3 \times 3$ and that the light is sliding through the image. With machine learning terms the flashlight is called a filter and the region that it is shining over is called receptive field. The filter also consists of an array of numbers, often referred as weights or parameters. An important detail is that the receptive field needs to have the same depth as the input, so for a $32 \times 32 \times 3$ image the filter is of size $3 \times 3 \times 3$. When the filter starts to convolve through the image it start at the top left with element-wise dot product, which adds up to 27 multiplications all summed up. This number only represent the top left corner in the image and by repeating this process by sliding the filter through the image a feature map is generated. The step size of each iteration can be adjusted and is called striding, for this example a stride of 1 is used. After the convolutional filter a feature map will have the size $30 \times 30 \times 1$, this is because there is only 900 unique locations a $3 \times 3$ kernel can fit into a $32 \times 32$ image. It is possible to get feature maps with the same size as the input by using a method called padding, which will be explained further in subsection 2.2.2.

In the case of more than 1 filter, which is usually the case, the feature maps are stacked in the third dimension, for example if a second filter is used the output would be $30 \times 30 \times 2$ [18]. An example of the whole process can be seen in Figure 2.3.

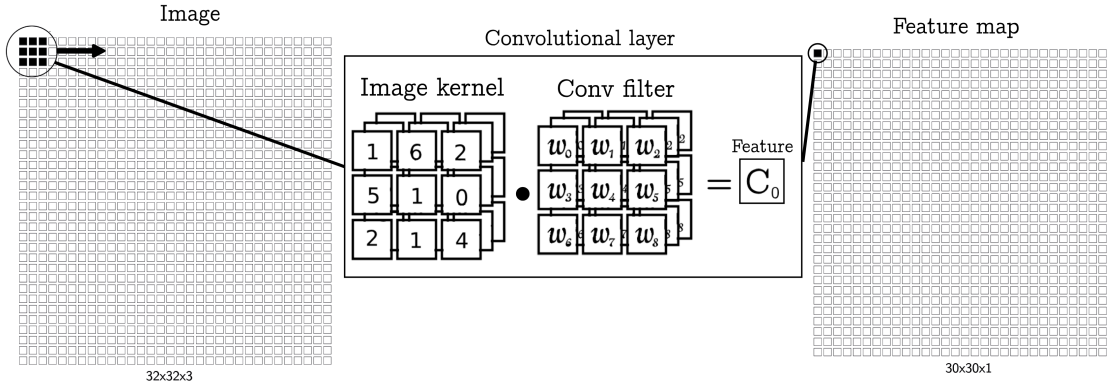



**Figure 2.3:** Illustration of $3 \times 3 \times 3$ convolutional filter applied to an $32 \times 32 \times 3$ RGB input image, the result is a feature map of size $30 \times 30 \times 1$

### 2.2.2 Padding operations

A commonly used operation in CNNs is padding. Padding is the process of adding extra pixels outside an image to make it larger. An example of a padding of size 1, i.e. adding a frame of extra pixels around the image, can be seen in Figure 2.4. The type of padding decides the values of the added pixel, in this image the type is called zero padding, meaning the added pixels have the value zero. Besides zero padding there exists for example mirrored and constant padding. In mirrored padding the added pixels are mirrored replicas of the image, this can be seen in Figure 2.5a. The constant padding type is very similar to zero padding, but instead of filling the

| | | | |
|---|---|---|---|
| 3 | 6 | 4 | 5 |
| 8 | 2 | 4 | 5 |
| 9 | 4 | 1 | 4 |
| 2 | 1 | 3 | 4 |

**(a)** Original image

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 3 | 6 | 4 | 5 | 0 |
| 0 | 8 | 2 | 4 | 5 | 0 |
| 0 | 9 | 4 | 1 | 4 | 0 |
| 0 | 2 | 1 | 3 | 4 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

**(b)** Padded image

**Figure 2.4:** An example of zero padding of size 1

added pixels with zeros it can be any constant value, an example of this can be seen in Figure 2.5b.

| | | | | | |
|---|---|---|---|---|---|
| 3 | 3 | 6 | 4 | 5 | 5 |
| 3 | 3 | 6 | 4 | 5 | 5 |
| 8 | 8 | 2 | 4 | 5 | 5 |
| 9 | 9 | 4 | 1 | 4 | 4 |
| 2 | 2 | 1 | 3 | 4 | 4 |
| 2 | 2 | 1 | 3 | 4 | 4 |

**(a)** Mirrored padding

| | | | | | |
|---|---|---|---|---|---|
| X | X | X | X | X | X |
| X | 3 | 6 | 4 | 5 | X |
| X | 8 | 2 | 4 | 5 | X |
| X | 9 | 4 | 1 | 4 | X |
| X | 2 | 1 | 3 | 4 | X |
| x | X | X | X | X | X |

**(b)** Constant padding, X can be any constant value.

**Figure 2.5:** Two padding examples with padding size 1. Original image can be seen in Figure 2.4a.

As described in subsection 2.2.1, when applying a convolutional filter with size $3 \times 3 \times 3$ to an image of size $32 \times 32 \times 3$, using stride 1, the output image will be of size $30 \times 30 \times 1$. However, it is not always desirable to get feature maps with a decreased size, this is were padding is useful. By adding a padding of size 1 before the convolution layer the feature map will keep the same size as the input. The relation between the input and output sizes for a convolutional layer can be described as,

$$y = \frac{x - K + 2P}{S} + 1, \tag{2.1}$$

where $y$ and $x$ are the input and output of height or width, $K$ is the filter size, $P$ padding size and $S$ stride.

Another reason to use padding is to attain the same amount of information from

every pixel. When applying a convolutional filter, the pixels further away from the border will be covered more times by the filter than the pixel around the edges, this means getting less information from the edges and corners of the image.

### 2.2.3 Activation functions

For a CNN to solve complex tasks it is not enough to only use convolutional layers, as the weights and biases in a convolutional layer transforms the input linearly. Linear equations are easy to solve but limited to the complexity of a problem, by using activation functions the input transformation will be non-linear resulting in a non-linear model.

The feature maps provided from filters, for example a convolutional filter, can be anything ranging from $-\infty$ to $+\infty$. By using an activation function the bounds of the values can also be controlled. Which neuron from the feature map that will be activated depends on which activation function that is used. The activation functions that will be discussed further in detail are Leaky ReLU and Softmax.

#### 2.2.3.1 Leaky ReLU

Leaky Rectified Linear Unit (Leaky ReLU) is a non-linear activation function that have been developed from the activation function Rectified Linear Unit (ReLU), the difference between the activation functions can be seen in figure 2.6.

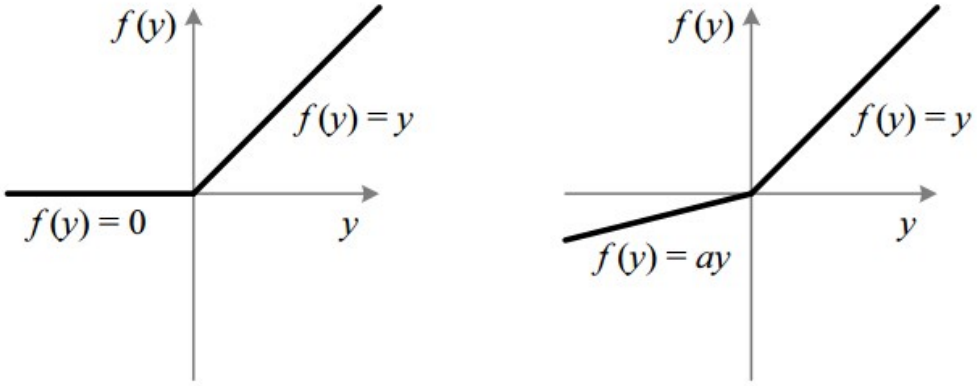



**Figure 2.6:** A comparison of the two activation functions ReLU (left) and Leaky ReLU (right).

ReLU have been a common activation function to use in the hidden layers because of its benefits. Firstly, the math is simple and the computational cost during training is low. Secondly, it is linear for positive values and thus will not plateau for large x. If an activation function plateaus it can lead to something called *The Vanishing gradient problem*, which can appear in the activation functions *Sigmoid* and *Tanh* [61]. During backpropagation every weight is updated proportional to the partial

derivative of the loss function $E_{loss}$ with respect to the weight. The problem with plateaus in activation functions is that it can lead to the gradient being vanishingly small which prevents the weights of being updated [61] .The ReLU function is defined as,

$$A(y) = max(0, y), \tag{2.2}$$

where y is the input to the activation function.

When using ReLU negative neurons will not be activated, this decreases the ability of the model to fit to data properly and information are lost [51]. This problem, called the *dying ReLU problem*, will not appear for Leaky ReLU as it pass negative values. The Leaky ReLU function follows as,

$$f(y) = \begin{cases} y, & \text{if } y \geq 0 \\ ay, & \text{if } y < 0, \end{cases} \tag{2.3}$$

where $a$ is a small constant which decide the impact of negative values. By using Leaky ReLU a network can keep the benefits from ReLU but also solve the *dying ReLU problem* at the same time with the cost of being a bit slower [79].

#### 2.2.3.2 Softmax

Softmax is an activation function that is usually used for the last layer in a classifier. In the example in Figure 2.2 the input to the Softmax is a feature map with the same size as the output, in this case a $2 \times 1$ array. Softmax then generate an output with values that represents probabilities for each class. The Softmax function follows as,

$$S(y_i)_i = \frac{e^{y_i}}{\sum e^{y_j}}, \tag{2.4}$$

where $y$ is the feature map from the previous layer and the indices $i$ and $j$ define the class.

The output from the Softmax function utilize Categorical Cross-Entropy as a loss function, the loss function will be discussed later in subsection 2.2.6.

### 2.2.4 Pooling operations: Max pooling

Pooling operations in CNNs are used for dimension reducing purpose, by reducing the sizes of the feature map the amount of operations is reduced and the efficiency increases. Similar to a convolutional filter, the pooling has a receptive field and a stride, and like the convolutional layer the pooling operation will stride through the feature map. The highest value in the receptive field will represent the whole area in the generated feature map.

Two common pooling operations are Average and Max pooling, Dominik Scherer *et al.* proved Max pooling to be more suitable for CNNss [70]. When V. Badrinarayanan *et al.* developed the architecture *SegNet* they introduced Max pooling

with indices, illustrated in figure 2.7. The difference is that the index of the max value in the receptive field is also saved, this index is then used in an operation called *upsampling*, this operation will be explained later in subsection 2.3.1.

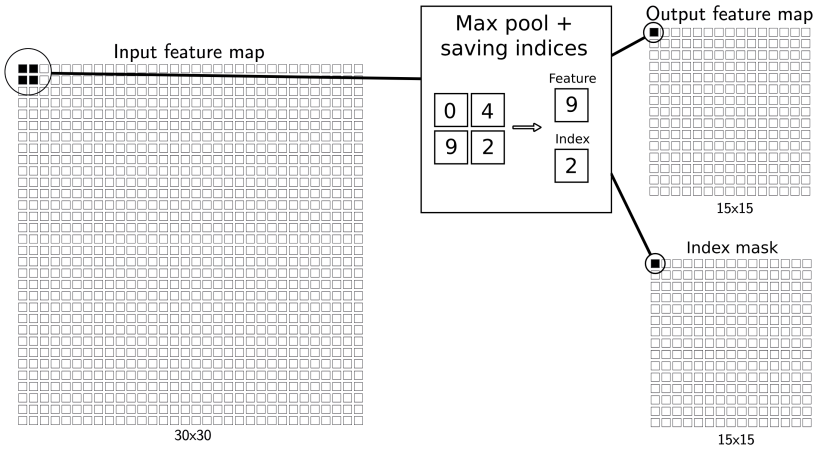



**Figure 2.7:** Illustration of max pooling operation with saving indices. The output is a feature map and corresponding index mask.

### 2.2.5 Supervised learning of a CNN

Supervised learning refers to that a model is trained using labeled data. The whole idea is to tune a model so it can map the correlation between the input and output data through iterations of parameter updates. To train a network using supervised learning, a dataset including both input and ground truth output data, is required.

To convert an input to a prediction the input has to go through the different layers and operations that exist in a CNN. When a prediction has been obtained it is then compared to the ground truth output with a loss function, $E_{loss}$, which is different depending on the network's task.

When the value from the loss function have been retrieved the goal is to either minimize or maximize this value. This means that all the parameters have to be tuned in order for the network to descend towards an optima. By calculating the partial derivative for each parameter the update direction can be obtained, this process is called backpropagation, the equation for backpropagation is,

$$\frac{\partial E_{loss}}{\partial w_i} \tag{2.5}$$

where $w_i$ is the parameter to be updated. Lastly, when the partial derivatives have been calculated, the parameters are updated such that the model can start to

converge to an optimum. A simple weight update could look like,

$$w_i = w_i - \eta \frac{\partial E_{loss}}{\partial w_i}. \tag{2.6}$$

The learning rate, $\eta$, will decide the size of the update step. The size of a learning rate can lead to different problems since it decides how rapid the network should adapt for each update. A high learning rate may descend too fast in wrong direction such that the model will end up with a suboptimal solution. A low learning rate risk to get stuck in a saddle point and never reach an optimum.

### 2.2.6 Loss function: Categorical Cross-Entropy

Categorical Cross-Entropy is a loss function commonly used for multi-class classifiers. The actual loss function is called Cross-Entropy but when it is combined with *Softmax* as activation function it is called Categorical Cross-Entropy. Its equation can be described as,

$$CE = -\sum_{i}^{C} t_i log(S(y)_i) \tag{2.7}$$

where $t_i$ is the ground truth, $C$ the number of classes and $S(y)_i$ is the prediction score for each class $i$. In multi-class classification the labels $t$ is one-hot encoded and as there is only one correct class, $t_p$, which alone will keep its prediction value.

### 2.2.7 Optimizer: Stochastic gradient decent

The loss function measure how wrong the predictions are, the optimizer use this information to make an educated update of the weights, with the goal to minimize the loss. When picking an optimizer there is two properties to consider, convergence rate and generalization, i.e. the performance on new data.

Stochastic gradient decent (SGD) is an optimizer that have been around since the 1950s and have the property to perform well on new data. The formula for the weight update using SGD is,

$$\theta \leftarrow \theta - \eta \nabla_\theta J(\theta; x^{(i)}; y^{(i)}), \tag{2.8}$$

where $\theta$ is the weights, $\eta$ the learning rate, $\nabla_\theta$ the gradient and $J$ the loss function. What differ SGD from other gradient decent variants is that SGD performs an update after each training sample that is randomly chosen.

The network can occasionally reach a local optima, that the network might assume is the global optima, which prevents the network to reach its fully potential. To make the network more likely to reach the global optima momentum can be used. Momentum is a method that accelerate the convergence in the right direction, it

will create a faster convergence and dampen the oscillations [66]. Momentum using the optimizer SGD is described in the two equations as,

$$\theta \leftarrow \theta - v_t, \tag{2.9}$$

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta; x^{(i)}; y^{(i)}), \tag{2.10}$$

where $v$ is the update term and $\gamma$ a constant that regulate the impact momentum have each update. As can be seen the momentum is depending on the past update term $v_{t-1}$.

### 2.2.8 Regularization: Batch normalization

Overfitting is a big problem within machine learning that occurs when a neural network descend to be too biased to the training data. This results in poor performance and high uncertainties when the model is exposed to data it has not seen before. To avoid this a bigger training set can be used, but as it can be hard and expensive to get more data, the problem can be solved by using regularizers instead. As regularizers are quite effective they are common to use within deep learning, one type of regularizer is *Batch normalization* (BN). BN was initially intended to solve the problem called covariate shift [37], which refers to when the training inputs and the test inputs have different probability distributions but the conditional distribution of the outputs remain the same [74], this frequently occurs when working with real world problems. To solve this problem BN was proposed, this method however was also proven to work well as a regularizer [38].

BN normalize the output from a previous layer over one mini-batch, it does this by subtracting the batch mean and divide it with the standard deviation of the batch. The equations for BN follow as,

$$h_i \leftarrow \mathbf{w}^\top \mathbf{x}_i \tag{2.11}$$

$$\mu_\beta \leftarrow \frac{1}{m} \sum_{i=1}^{m} h_i \tag{2.12}$$

$$\sigma_\beta^2 \leftarrow \frac{1}{m} \sum_{i}^{m} (h_i - \mu_\beta)^2 \tag{2.13}$$

$$\hat{h_i} \leftarrow \frac{h_i - \mu_\beta}{\sqrt{\sigma_\beta^2 + \epsilon}} \tag{2.14}$$

$$y_i \leftarrow g(\gamma \hat{h}_i + \beta) = g(\mathrm{BN}_{\gamma,\beta}(h_i)), \tag{2.15}$$

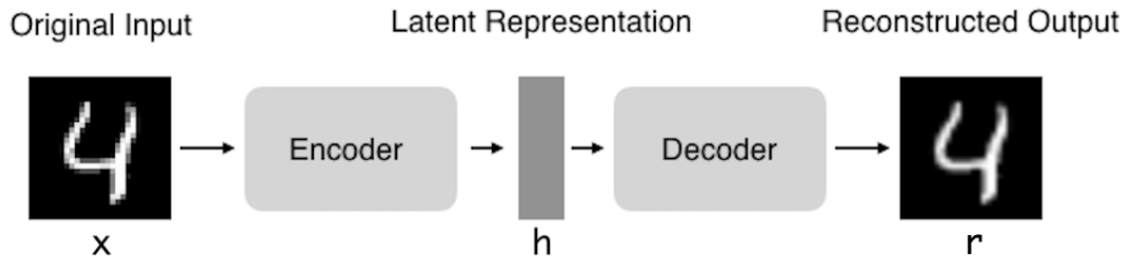



**Figure 2.8:** An Autoencoder for image retrieval, adapted from [36]

where $h$ and $\hat{h}$ are the hidden values before and after the BN. $\mathbf{w}$ is the weights, $\mathbf{x}_i$ the input and $g$ the activation function. $\mu_\beta$ is the mean and $\sigma_\beta^2$ variance of the mini-batch. The parameters $\gamma$ and $\beta$ are the scale variable respectively shift variable and are learned in the optimization process [38, 37].

By using BN the network sees the training images in each mini-batch in a conjunction together, this removes the deterministic values that would otherwise exist for each training image [38].

## 2.3 Autoencoder

An autoencoder is a deep neural network with the purpose to compress (encode) data into a short string of code that acts as a fingerprint towards the input which is then extracted (decoded) to a target output. By forcing data dimension reduction, it limits which features the network will act upon, therefore features that is inconsistent and not frequently appearing in the dataset will disappear when the data is encoded, which means that noise and unwanted data will be filtered out in the encoder. An autoencoder can be used to retrieve damaged or corrupted images, an example of a flowchart trained on MNIST can be seen in Figure 2.8. In this example the encoder compresses the input data and can be represented as an encoder function $h = f(x)$, where $h$ is called the latent representation which is the string of code. The decoder that have the same architecture as the encoder but reversed, can be represented as a decoder function $r = g(h)$. It is an unsupervised learning algorithm, meaning that no labeled data is required to train the model and the parameters are learned through backpropagation where the target values are set to be equal to the input data, $x = y$.

Autoencoders have also been utilized to pretrain classification models. Simply put, the model is first trained on a large unlabeled dataset, then the weights are saved and the decoder is replaced with a couple of fully connected layers and then tuned on a small labeled dataset. Another classification problem within CV where autoencoders are used is semantic segmentation, which will be discussed further in section 2.4. Compared to the other problems this requires labeled data to train the model which make it to a supervised learning method. In Figure 2.9 a fully convolutional encoder-decoder architecture for semantic segmentation is illustrated as an example.

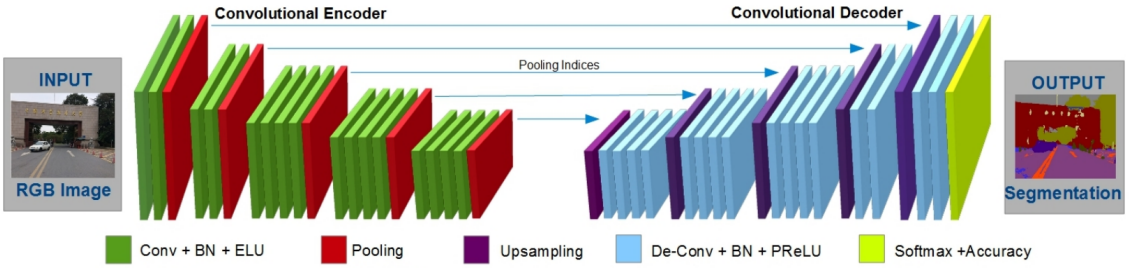



**Figure 2.9:** An example of a fully convolutional autoencoder architecture for image retrieval that restores damaged or corrupt images, source from [80]

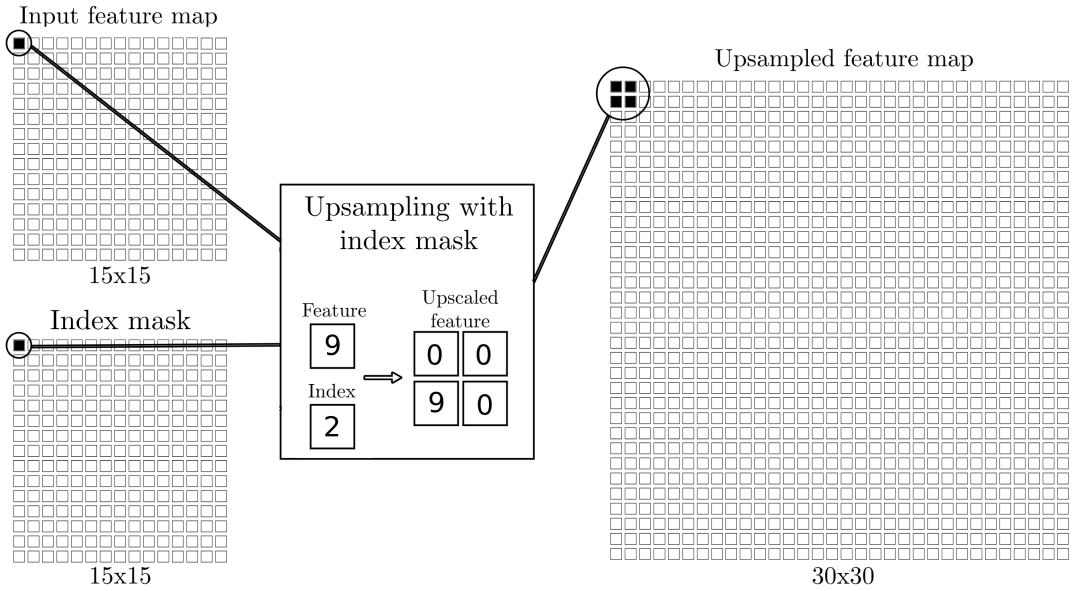



**Figure 2.10:** Illustration of an upsampling operation with saved indices from Figure 2.7

### 2.3.1 Upsampling in decoder

Models with an autoencoder architecture have to upscale the feature maps in the decoder in order to output an equally large prediction as the input size. Compared to Max pooling, which reduced the dimensions, the operation called upsampling does the opposite and increases the dimensions. Although, there are several different upsampling methods and to mention a few as example; *Transposed convolution* or *Bilinear upsampling* [78]. V. Badrinarayanan *et al.* upsampled their features by using the corresponding indices from the Max pooling operations in the encoder [5], the operation can be seen in Figure 2.10.

## 2.4 Semantic segmentation

Semantic segmentation is a rather advance classification task within computer vision. The goal of semantic segmentation is to classify regions in the image with its corresponding class. An example of the input and the corresponding output of such a network is shown in Figure 2.11. The left image is the input image to the network and the right image is the semantically segmented image. Each color in the output image represents a specific class.

**(a)** Original image **(b)** Pixel-wise classified image

**Figure 2.11:** An example of an input and output of an semantic segmentation network. a) Original input image, b) the pixel-wise classified output. Image from KITTI dataset [28]

### 2.4.1 SegNet

SegNet is a deep fully convolutional neural network for semantic segmentation [5]. The architectures consists of an encoder, a corresponding decoder followed by an output layer with Softmax as the activation function, illustrated in Figure 2.12. The network VGG16 designed for image-level classification is used as the encoder and reversed for the decoder. Although, SegNet only use the 13 first layers and the last three fully connected layers are excluded. The size of the convolutional layers in each block is $3 \times 3$ and the Max pooling is $2 \times 2$ with stride 2. The amount of extracted feature maps changes for each block and is respectively, 64, 128, 256, 512 and 512 [73]. In the convolutions the feature maps is batch normalized and processed through a ReLU as the activation function. The decoder has a similar process but in reverse and up-sampling instead of Max pooling [5].

Another feature in SegNet is the use of pooling indices during decoding which is saved from the Max pooling during encoding. This make it possible to perform non-linear upsampling and result in a feature map of higher resolution [5] .

## 2.5 Unsupervised depth estimation

Depth estimation is an important step toward situational awareness and scene understanding. The goal is to output a pixel-wise depth map of a given monocular or

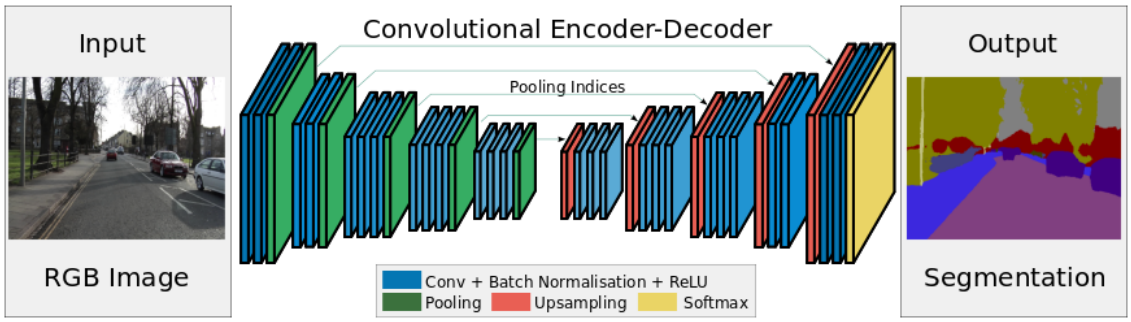



**Figure 2.12:** Illustration of the SegNet architecture that is designed to take a RGB image as input and semantic segmentation as output. VGG16 is used as the backbone architecture. Adapted from [5]

stereo-pair image, this is done by assigning a value representing the depth to each pixel in the image. An example of a depth estimation provided from a model can be seen in Figure 2.13.

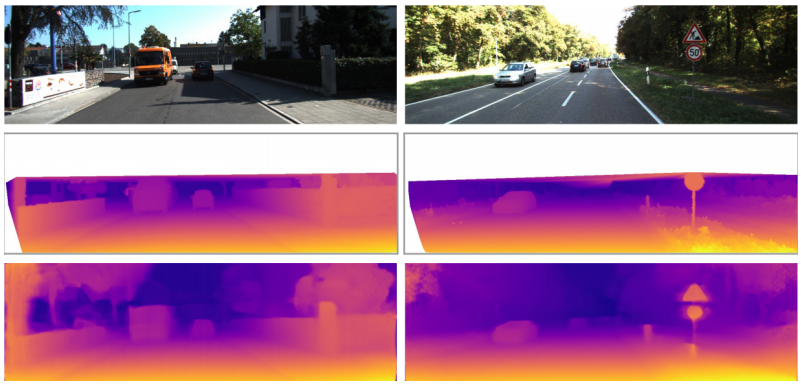


**Figure 2.13:** An example of a depth estimation. The top to bottom: input image, ground truth disparities and estimated depth disparities. Adapted from C. Godard *et al.* [30].

There are depth estimation models that are either trained through supervised [21, 44] or unsupervised [30, 64] learning. The advantage with unsupervised learning is that no labeled data is required and is therefore not restricted to specific data scenes where large image with corresponding depth map are available [30].

### 2.5.1 PyD-Net

The CNN PyD-Net is an example of an unsupervised depth estimation network [64]. The network is designed to only process one image, but the training framework use stereo pair images during training, the reason lies behind the training framework and, a simple illustration of the training framework is shown in Figure 2.14 [64].

PyD-Net was proposed to deliver good depth estimations but with a simple structure such that it could be implemented for real-time use [64]. An illustration of the

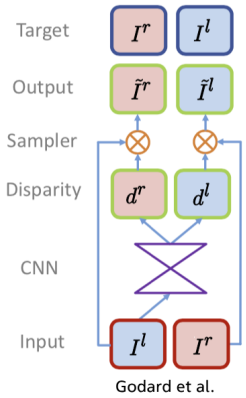



**Figure 2.14:** A simple illustration of the training framework used for PyD-Net, adapted from [30]. The CNN, PyD-Net, is fed with the left image and decodes left and right depth images. By sampling the left image with the predicted right depth image the right image can be retrieved and vice versa for the left depth image. The retrieved images are then compared to the target images that are the original stereo pair.

architecture can be seen in Figure 2.15. The input features are extracted by an encoder made of 12 convolutional networks stacked in 6 different levels, L1 to L6. Each level in the encoder starts with a convolutional layer with stride 2 followed by another convolutional layer with stride 1, both with the size $3 \times 3$ and with a Leaky-ReLU with $\alpha = 0.2$. The number of extracted features increases for each downsampling module, respectively 16, 32, 64, 96, 128 and 192. The resolution of the image is divided by 2 for each level, starting from $\frac{1}{2}$ in L1 to $\frac{1}{64}$ in L6. The extracted features from the encoder is initially processed in the highest level (L6) by a depth decoder made of 4 convolutional layers which produce 96, 64, 32 and 8 feature maps respectively. The extracted output is later used twice. Firstly, to extract a depth map for the current resolution by the use of a Sigmoid function, and secondly, to pass information to the lower level in the pyramid. In the lower level (L5) the encoded features is concatenated with the decoded features from the higher level (L6) that is upscaled with a deconvolution layer with stride 2. The concatenated features are then processed through the same process as the decoder in L6. This process is repeated all the way down to the first level with the highest resolution. Similar to the encoder the convolutions use $3 \times 3$ kernels with Leaky ReLU except the last layer which uses Sigmoid to normalize the outputs. [64]

## 2.6 Public datasets

The demand of qualitative data has increased along the development of machine learning. For supervised learning the performance of a model depends on architecture, training methods and dataset. The purpose with a supervised network is to design a network which can predict a possible output given an input, which is all

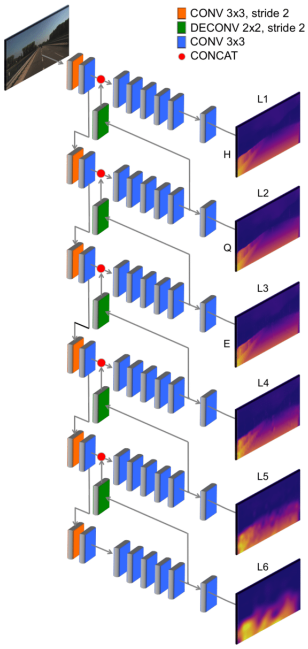



**Figure 2.15:** Illustration of the PyD-Net architecture, the input is an RGB-image and output six disparity maps with different feature levels. Source: Adapted from [64].

based on a likelihood regarding what the model has seen during training. It is therefore important to know which data a model should be exposed to during training. As an example, for image recognition in urban areas it is often preferable to have a dataset with a variation of images with different environments.

There are several public datasets with some difference in size, labels, cities, weather and daytime. The datasets KITTI [28, 52, 53, 1], Cityscapes [15] and BDD100K [81] are three datasets focused on urban environments and are commonly used in CV for autonomous drive. These datasets will be described in detail in this chapter.

### 2.6.1 Cityscapes

The purpose of Cityscapes is to benchmark networks which focus on semantic information (pixel-level, instance-level, and panoptic) of urban street scenes. It contains images from 50 different cities captured during several months, on different times of the day and in different weather conditions. There is a total of 3178 fine annotated images and 20000 coarse annotations, all with a resolution of $2048 \times 1024$. The different annotations is semantic, instance-wise and dense pixel annotations [15].

Table 2.1 displays the percentage of pixels for each class in Cityscapes. Classes like *Road*, *Building*, and *Vegetation* are well presented in the dataset, while the *Motorcycle* class is especially underrepresented.

**Table 2.1:** The percentage of pixel representation of each class present in the Cityscapes dataset.

| | | | | | |
|---|---|---|---|---|---|
| **Void** | 11.472 | **Traffic light** | 0.184 | **Car** | 6.192 |
| **Road** | 32.640 | **Traffic sign** | 0.488 | **Truck** | 0.237 |
| **Side-walk** | 5.387 | **Vegetation** | 14.101 | **Bus** | 0.208 |
| **Building** | 20.206 | **Terrain** | 1.025 | **Train** | 0.206 |
| **Wall** | 0.580 | **Sky** | 3.558 | **Motorcycle** | 0.087 |
| **Fence** | 0.777 | **Person** | 1.079 | **Bicycle** | 0.366 |
| **Pole** | 1.087 | **Rider** | 0.120 | | |

### 2.6.2 KITTI

KITTI was developed to create benchmarks for networks with tasks such as stereo, optical flow, visual odometry, 3D tracking and 3D object detection. Images are captured from the city of Karlsruhe, in rural areas and on highways. The stereo and flow matching dataset that was generated in 2012, contains 194 training and 195 test image pairs at a resolution of 1240 x 376 pixels [28]. Whereas the stereo and flow dataset from 2015, consist of 200 training and 200 test images [52, 53]. KITTI later provided a benchmark for semantic segmentation which contains semantic annotated images taken from the stereo and flow dataset 2015. The data format and metrics are in conform with Cityscapes [1].

The Table 2.2 shows the representation of classes in KITTI. The classes *Road* and *Vegetation* is especially well represented. The classes *Terrain* and *Sky* appears relatively often compared to Cityscapes.

**Table 2.2:** The percentage of pixel representation of each class present in the KITTI dataset

| | | | | | |
|---|---|---|---|---|---|
| **Void** | 3.836 | **Traffic light** | 0.307 | **Car** | 6.01 |
| **Road** | 23.098 | **Traffic sign** | 0.551 | **Truck** | 0.224 |
| **Side-walk** | 3.824 | **Vegetation** | 30.323 | **Bus** | 0.066 |
| **Building** | 8.156 | **Terrain** | 9.42 | **Train** | 0.215 |
| **Wall** | 0.919 | **Sky** | 10.668 | **Motorcycle** | 0.009 |
| **Fence** | 0.835 | **Person** | 0.094 | **Bicycle** | 0.058 |
| **Pole** | 1.360 | **Rider** | 0.028 | | |

### 2.6.3 BDD100K

BDD100K is the largest public dataset available for autonomous driving (AD) purposes. By introducing new tools to annotate data such as image-level tagging, bounding box and polygon annotation they managed to develop a diverse public dataset containing 100K video clips, mainly captured within USA. The dataset contains annotated data such as object detection, lane detection, drivable area and semantic segmentation. [81]

Table 2.3 shows the class representation in BDD100K. Similar to Cityscapes the classes *Road*, *Building*, and *Vegetation* appear often, however in this dataset the classes *Void* and *Sky* is common as well and the classes *Train*, *Motorcycle* and *Bicycle* are poorly represented.

**Table 2.3:** The percentage of pixel representation of each class present in the BDD100K dataset

| | | | | | |
|---|---|---|---|---|---|
| **Void** | 18.720 | **Traffic light** | 0.180 | **Car** | 8.114 |
| **Road** | 21.493 | **Traffic sign** | 0.339 | **Truck** | 0.971 |
| **Side-walk** | 2.035 | **Vegetation** | 13.207 | **Bus** | 0.556 |
| **Building** | 13.259 | **Terrain** | 1.031 | **Train** | 0.014 |
| **Wall** | 0.479 | **Sky** | 17.299 | **Motorcycle** | 0.024 |
| **Fence** | 1.031 | **Person** | 0.251 | **Bicycle** | 0.051 |
| **Pole** | 0.924 | **Rider** | 0.021 | | |

## 2.7 Performance evaluation metrics for semantic segmentation

How a network performs on a desired task, e.g. classify or detect objects in an image, can be calculated using different evaluation metrics. These metrics focus differ, some focus on the ratio of correct estimated pixels were other focus on evaluating the accuracy between classes. In this chapter we will discuss two kinds of performance evaluation metrics for semantic segmentation, *accuracy* and *confusion matrix*.

### 2.7.1 Accuracy

Most papers use accuracy to measure the performance of a network, it is easy to implement as it is a built-in metric in many of the commonly used functions within machine learning. There exist various accuracy metrics for different network tasks, but also different metrics within each specific task.

The paper written by A. Garcia-Garcia *et al.* proposes four metrics to calculate accuracy for semantic segmentation [27]. To describe the metrics, the variables $C_{ii}$, $C_{ij}$ and $C_{ji}$ is used. $C_{ii}$ defines the so called true positives which are the correct classified pixels, $C_{ij}$ and $C_{ji}$ symbolize the false positive respectively false negative. The metrics are defined as:

$$PA = \frac{\sum_{i=0}^{k} C_{ii}}{\sum_{i=0}^{k}\sum_{j=0}^{k} C_{ij}}. \tag{2.16}$$

This is a very simple method which gives a general overview of how good the network is. However, it do not give any information on which of the classes the network performs well.

**Mean Pixel Accuracy (MPA)** calculate the per-class pixel accuracy, which is then averaged over the number of classes, as

$$MPA = \frac{1}{k+1} \sum_{i=0}^{k} \frac{C_{ii}}{\sum_{j=0}^{k} C_{ij}}. \tag{2.17}$$

This metric gives an overview of which classes the network performs well.

**Mean Intersection over Union (MIoU)** represent how much the ground truth object overlaps with the predicted object. IoU, also called Jaccard Index, describes the ratio between the intersection over the union of the predicted segments and the ground truth [16]. This is calculated for each class and thereafter averaged to get the mean intersection over the unions, as

$$MIoU = \frac{1}{k+1} \sum_{i=0}^{k} \frac{C_{ii}}{\sum_{j=0}^{k} C_{ij} + \sum_{j=0}^{k} C_{ji} - C_{ii}}. \tag{2.18}$$

This method is the most commonly used for computing segmentation accuracy and is used in various competitions, like the PASCAL VOC and KITTI challenge [22][2].

**Frequency Weighted Intersection over Union (FWIoU)** is an extension of MIoU. The FWIoU takes into account how frequent a class appear, this data is then used to weight the class significance, as

$$FWIoU = \frac{1}{\sum_{i=0}^{k} \sum_{j=0}^{k} C_{ij}} \sum_{i=0}^{k} \frac{\sum_{j=0}^{k} C_{ij} C_{ii}}{\sum_{j=0}^{k} C_{ij} + \sum_{j=0}^{k} C_{ji} - C_{ii}}. \tag{2.19}$$

### 2.7.2 Confusion matrix

A confusion matrix is a common way to show the performance of a network for each individual class. It is a method to show for which classes the network is confused and mixing up during prediction. The columns in the matrix represent the predicted classes, the rows the actual classes and the diagonal the correctly classified samples for each class.

In Table 2.4 a simple example of a confusion matrix is visualized. By looking at the row for class A it can be established that there are in total 20 class A samples. 14 of them were correctly classified, out of the 6 that were miss-classified, 4 of them

were classified as class B and 2 of them as class C. The same rules apply for class B and C.

**Table 2.4:** Example 1 of a confusion matrix, values represent the numbers of instances.

| | | **Predicted class** | | |
|---|---|---|---|---|
| | | **Class A** | **Class B** | **Class C** |
| **Actual class** | **Class A** | **14** | 4 | 2 |
| | **Class B** | 6 | **5** | 9 |
| | **Class C** | 2 | 0 | **18** |

A confusion matrix can also be used to show the performance of a semantic segmentation model. The confusion matrix then includes the prediction for every single pixel in the dataset. However, this quickly becomes a large number. To make the result more intuitive the values can be shown as percentage [47]. An example of this is shown in Table 2.5, where 70 % of the pixels for class A were correctly classified, 20 % were miss-classified as class B, and 10 % as class C.

**Table 2.5:** Example 2 of a confusion matrix, values is represented in percentage.

| | | **Predicted class** | | |
|---|---|---|---|---|
| | | **Class A** | **Class B** | **Class C** |
| **Actual class** | **Class A** | **70** % | 20 % | 10 % |
| | **Class B** | 20 % | **50** % | 30 % |
| | **Class C** | 10 % | 0 % | **90** % |

## 2.8 Robustness

The term *Robustness* appears quite often in articles within machine learning regarding the performance of a model. However, when discussing what a robust model is the definition vary a lot. In this chapter different definitions are discussed to get a sense on what definitions that exists in the CV field today.

When some researchers discuss robustness in their papers, they do not specifically define it. By reading their articles you can understand that they define it as how well a network performs when tested on a similar but independent dataset from the one it was trained on [32]. This is often tested by comparing the test error with the training error, this type of robustness will be called *test set robustness* in this project.

Another definition of robustness that has become more common since 2015 for CNNs is susceptibility to adversarial input perturbations. In 2015 I. J. Goodfellow *et al.* noticed that CNN classifiers could be easily tricked with carefully constructed noise [31]. An adversarial input is an input an attacker has intentionally designed to cause the network to give an incorrect prediction with high confidence. The attack can be

either physical or digital. A physical attack can for example be a carefully designed sticker on a road sign while a digital attack is created by changing the input to a network by altering specific pixels leaving the input in many cases nearly unchanged for a human eye [24, 7]. Adversarial perturbations can be applied to networks with tasks such as image recognition, speech recognition, and semantic segmentation [31, 9, 3]. What seems to be the most common way to measure this type of robustness is the percentage of unsuccessful adversarial attacks against the network. There exist challenges like NIPS Adversarial Vision Challenge to compete and measure the robustness against adversarial attacks [72]. There is no general defence system that handle all kinds of adversarial attacks, however a network can be trained to be resistant to one type of attack, for example, against a first-order adversary attacks [55]. Some networks architectures is inherently more robust against attacks than others, an example of this is networks that have residual connections and performs multiscale processing [3].

In image recognition people have managed to create physical stickers to make networks predict an incorrect answer, however this is not as easily accomplished for semantic segmentation. A paper about adversarial attacks against semantic segmentation network presented a method to remove whole classes predicted from the network [54]. Though, this was done by carefully and precisely redesign the digital input and they did not manage to create any physical examples. It is more difficult to create physical attacks, this is because the adversarial example needs to be able to work from different ranges, light conditions, camera viewpoints etc [4].

When discussing adversarial attacks, we refers to a worst-case scenario for the network. Instead of evaluating against these extreme perturbations some researchers instead evaluate against noise. Alhussein Fawzi *et al.* [25] have shown how well a network performs while the input is affected by random, and semi-random noise. The noise robustness of the networks is tested in a similar manner as for adversarial perturbations. Another approach by Devrim Unay *et al.* [20] proposed a method to create a robust network for identifying neurodegenerative diseases in Magnetic Resonance (MR) brain images . To test the robustness of the network they added noise in the form of different intensity filters to the input image. In these two examples different kind of noise were added to the input of the network.

# 3

# Methods

The aim of this project was to study how a fusion of two deep CNNs would affect the performance for different tasks. Although, a limitation was to only analyze and test on one of the tasks, the pixel-wise classification. To reach a conclusion the methodology illustrated in Figure 3.1 was applied. The process consists of six steps whereas A-C is preprocessing, D the main task and E-F is analysis, each step are explained in detail further in this chapter.

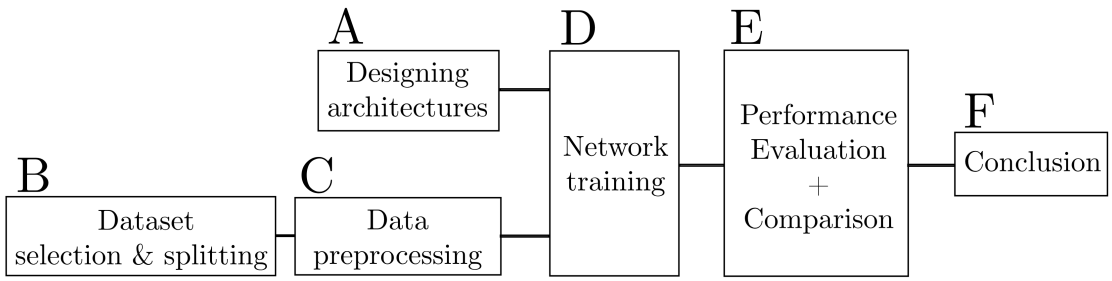



**Figure 3.1:** Flowchart of this project methodology, consisting of six steps with the categories A-C preprocessing, D main task and E-F analysis.

## 3.1 Deep-learning architecture for semantic segmentation

Architectures for different networks in this project follows some similar boundaries. Firstly, the same design as SegNet were applied for the autoencoder that process the RGB input image. Secondly, each network have a RGB and a depth image as input and output pixel-wise classification. The depth images was predicted from PyD-Net with pre-trained weights from the original paper [64]. This let us keep the uncertainties from PyD-Net which is a closer simulation of a fusion between two CNNs.

Lastly, they use the same CNN operations but varies in how they are applied. The convolutional layers are of size $3 \times 3$ with stride 1 except the last one before the Softmax activation function which is of size $1 \times 1$. In every hidden layer the convolved

feature maps are also batch normalized and thereafter non-linearly transformed through Leaky ReLU with $\alpha = 0.1$. The size of the Max pooling layer is $2 \times 2$ with stride 2. The upsampling factor is $2 \times 2$ with stride 2 and is upsampled with the pooling-indices that is saved from the max-pooling layer. Similar to SegNet the proposed architectures also have five different blocks with different number of filters in each convolutional layer. From large to small blocks the amount of filters is 64, 128, 256, 512 and 512. The last layer contains equally amount of filters as classes, in our case 10.

### 3.1.1 Architecture 1 - P-SegNet

The first architecture that can be seen in Figure 3.2 almost have an identical architecture to SegNet in Figure 2.12. The only difference is that the RGB-image is processed through PyD-Net first to produce a depth image and then concatenated with the RGB-image, thus it got its name PyD-SegNet (P-SegNet). The input size to the autoencoder becomes $W \times H \times 4$, and because of the fourth channel the number of parameters in the first convolutional channel will be larger than for SegNet. The idea of this architecture came from the RGB-D images used to create the FCN by J. Long *et al.* [50].

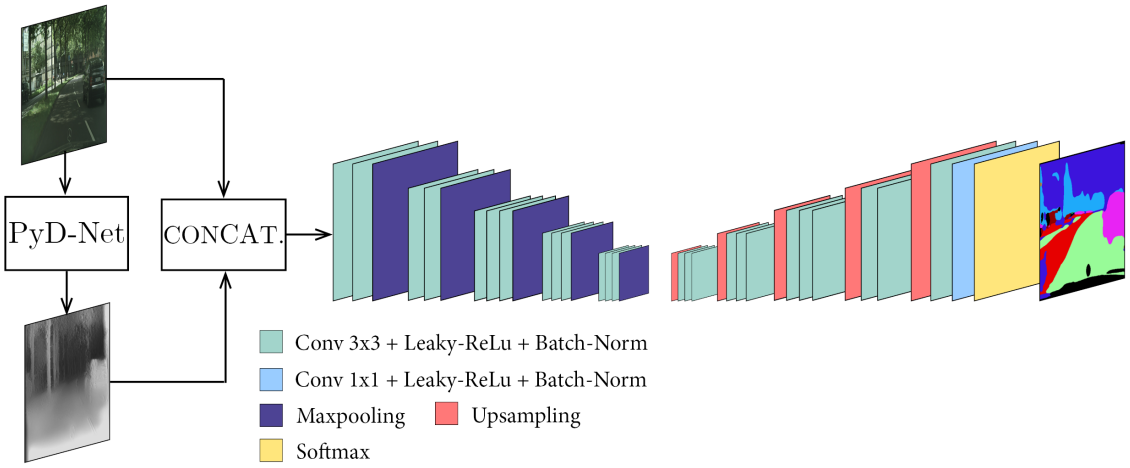



**Figure 3.2:** P-SegNet architecture

### 3.1.2 Architecture 2 - DF-SegNet

Another approach to input the depth image is to concatenate it in the last block after the upsampling layer and use it as a decoded feature map, thus it is called Depth Feature SegNet (DF-SegNet). The same layers as in SegNet were used, the design can be seen in Figure 3.3.

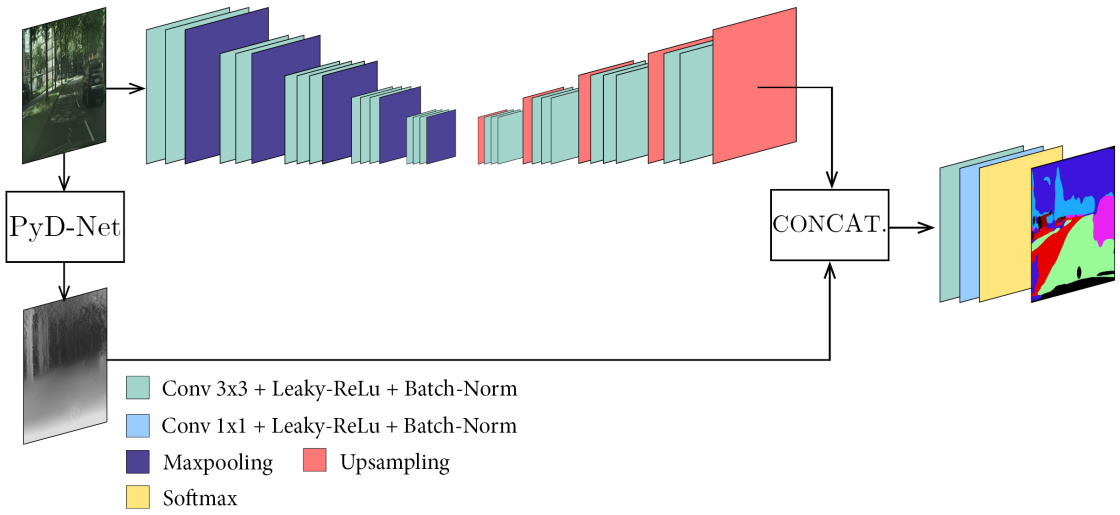



**Figure 3.3:** DF-SegNet architecture

### 3.1.3 Architecture 3 - D-SegNet

Double SegNet (D-SegNet) consist of two Autoencoders to process the depth and RGB image separately and are concatenated right before the last convolutional layer. The architecture can be seen in Figure 3.4.

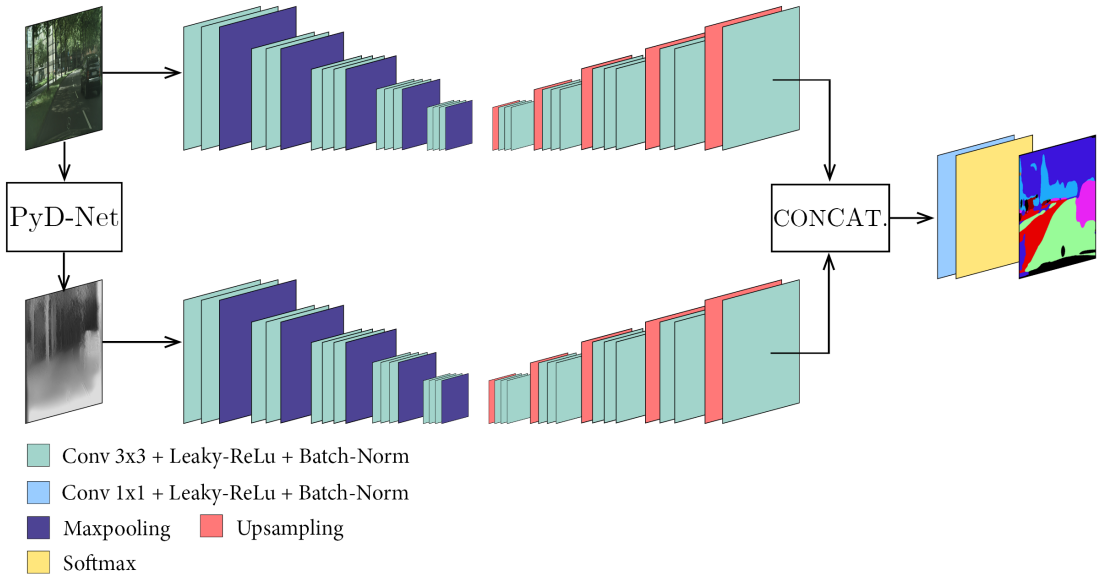



**Figure 3.4:** D-SegNet architecture

### 3.1.4 Architecture 4 - BD-SegNet

Basic Double SegNet (BD-SegNet) is quite similar to D-SegNet. The only difference is that the autoencoder that process the depth image is less complex. The architecture can be seen in Figure 3.5.

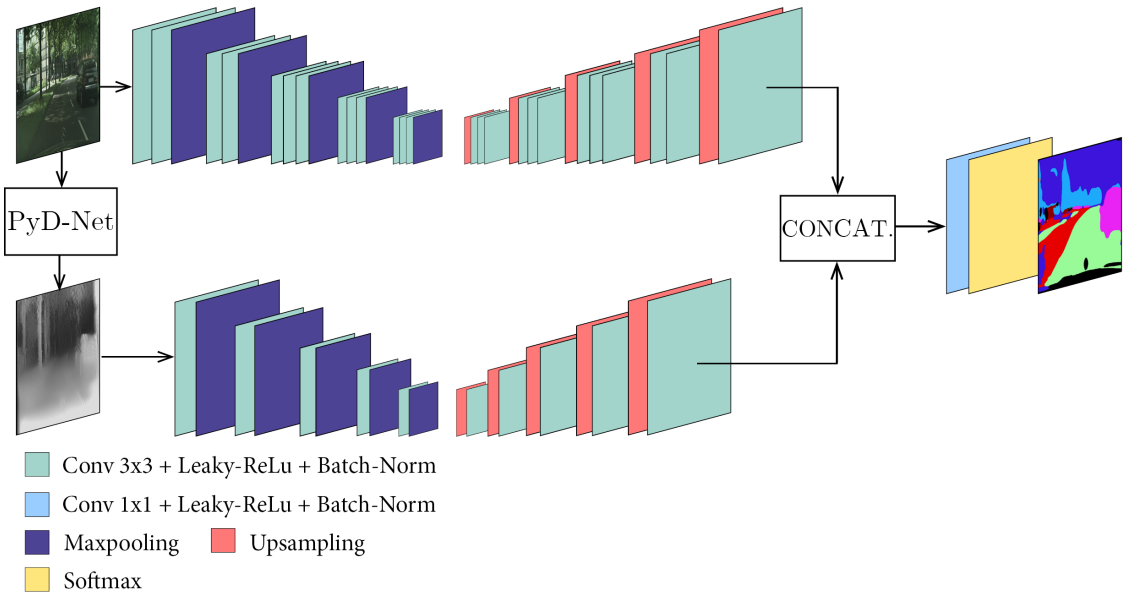



**Figure 3.5:** BD-SegNet architecture

### 3.1.5 Architecture 5 - EF-SegNet

The last architecture we designed was called Encoder Fuse SegNet (EF-SegNet). In this architecture we fed the RGB and depth image into two separate encoders with the same characteristics. The output layers from both encoders were then concatenated before they were fed into a shared decoder.

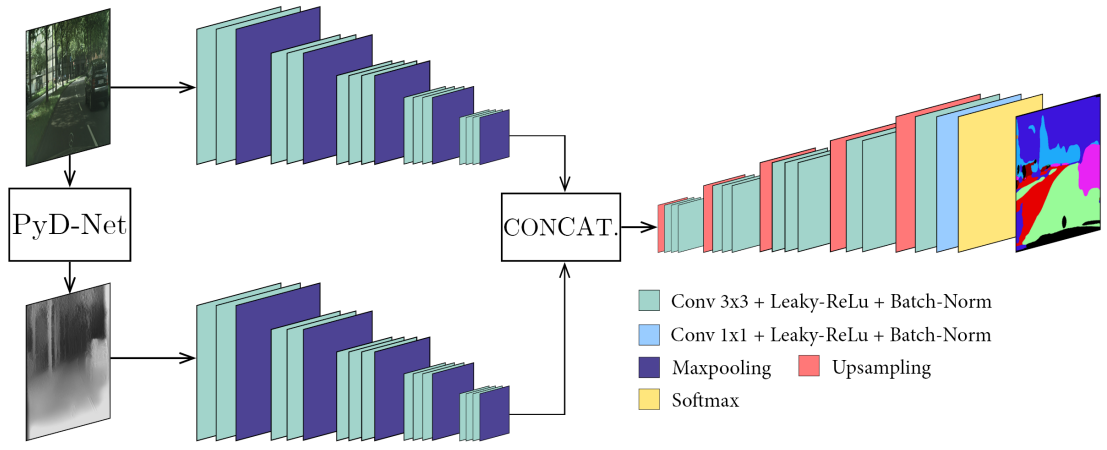



**Figure 3.6:** EF-SegNet architecture

In practice this means that the network deconstructs the features from the images separately and then tries to reconstruct based on the feature maps. There is also not equally amount of Max pooling layers as upsampling layers which there have been in the other architectures. In EF-SegNet the upsampling layers only use the Max pooling indices from the RGB-image encoder. The architecture for EF-SegNet can be seen in Figure 3.6.

## 3.2 Dataset

Since the task semantic segmentation with supervised learning is analyzed the data that is exposed to the networks have a huge impact on the outcome. First the dataset has to meet the requirements for the specific task. Secondly, the variation in the dataset have to be good enough such that there are enough differences in the validation and test set to be able to evaluate how the network performs in general.

### 3.2.1 Dataset selection

The quality of the data used to train a neural network can have major impact of the network performance. The data should be a good representation of the reality and the number of samples and the variation in the dataset should be large enough to increase the network's ability to be generalized.

As it is supervised semantic segmentation networks that is trained the dataset needs to contain pixel-level annotations, this is therefor our first requirement. This project focus on autonomous driving, thus the data need to contain road scenes. These are the only two requirements for the dataset, however it is preferable for the dataset to contain stereo vision and annotated depth maps. Stereo vision is preferred to give the opportunity to retrain the PyD-Net network in future work. Ground truth depth information could, also in future work, replace the depth data from PyD-Net to see how the semantic segmentation change in performance compared to having the PyD-Net prediction as input to the network.

Another preferred property of the dataset is to have benchmark results for other networks. Then the networks in this project can be compared to other networks and a better understanding of the network's performance can be gained.

The datasets discussed in section 2.6 was KITTI, Cityscapes and BDD100k. All of them have pixel-level annotations and images taken in an urban environment and therefore meets the two requirements. All of them have stereo vision however only KITTI have depth information. KITTI and Cityscapes is the most commonly used dataset by the three and have benchmark results for a range of networks [27]. KITTI have 200 training images with pixel-annotations, Cityscapes 3178 images and BDD100K 10 000 images.

All of the datasets exceed in one of the categories mentioned above. Though, we

decided to use Cityscapes and KITTI in this project. The reason BDD100k was not chosen was because it do not have a lot of benchmark results and that Cityscapes was large enough. Training the networks on both Cityscapes and BDD100k would be too overwhelming for this study.

In the training session we only used Cityscapes as a training dataset. The reason why we did not use KITTI as a training set was because of the small sample size of the dataset. Instead KITTI was used as a test set.

### 3.2.2 Dataset splitting

During development of a network in deep learning it is beneficial to split the dataset to three different sets, training, validation and test set. In total Cityscapes have 3178 fine annotated images with pixel-wise classification. These images have been recorded over 21 different cities. To reduce that the network would end up bias towards the different cities the dataset was splitted between the cities. The sets were splitted such that the training set contained of 17 cities and both the validation and test set contained of 2 cities each. Each city sequence contained of approximately 150 images each.

To ensure that the methodology would be consistent with the results the dataset was splitted differently for three rounds. The different sets, train, validation and test, had the same number of cities as previously mentioned but which cities were different for every round and randomly chosen. The presence of the classes and the number of images for the rounds can be seen in Table 3.1. The idea with the table is to show the data that the networks have been trained on for the different rounds, which mean that the values are taken from data that is exposed to the networks.

**Table 3.1:** The differences of the training sets between the rounds after the data preprocessing; resizing, class selection and shuffle. The values for the classes are how frequent they appear in the dataset in percentage.

| | | **Round** | | |
|---|---|---|---|---|
| | | **1** | **2** | **3** |
| | **Images** | **2661** | **2560** | **2596** |
| **Class presence [%]** | Void | 15.33 | 14.82 | 15.53 |
| | Road | 32.61 | 32.47 | 32.53 |
| | Sidewalk | 5.29 | 5.33 | 5.43 |
| | Building | 19.60 | 20.15 | 20.26 |
| | Pole | 1.15 | 1.05 | 1.09 |
| | Vegetation | 14.53 | 14.44 | 13.71 |
| | Terrain | 0.97 | 1.00 | 0.89 |
| | Sky | 3.42 | 3.53 | 3.46 |
| | Person | 1.10 | 1.06 | 1.14 |
| | Car | 6.00 | 6.16 | 5.97 |

## 3.3 Data pre-processing

Before the data was used to train the networks, it had to be processed to be compatible with the training framework. This chapter explains the selection of classes, one-hot encoding, resizing, generation of depth-predictions and normalization.

### 3.3.1 Selection of classes

Originally, the Cityscapes dataset contained of 33 different labeled classes. As seen in Table 2.1 the proportions of how present each class is in the dataset variate. Therefore, it was decided not to include every single class and only consider the classes that had more than 1% presence in the dataset. The classes which were excluded were merged with the class *Void* instead. The classes that we used can be seen in Table 3.2.

**Table 3.2:** The classes used from Cityscapes in this project

| # | **Name** |
|---|---|
| 0 | Void |
| 1 | Road |
| 2 | Sidewalk |
| 3 | Building |
| 4 | Pole |
| 5 | Vegetation |
| 6 | Terrain |
| 7 | Sky |
| 8 | Person |
| 9 | Car |

### 3.3.2 One-hot encoding

Before initialization of the training session the data had to be preprocessed such that Categorical Cross-Entropy could be utilized as a loss function. The labeled data had annotations for 10 different classes for every pixel and had to be remade to one hot-encoded matrices. This was done by stacking 10 layers with the same height and width as the image, which mean that the size of the matrix was $H \times W \times N$ which contained a value of 1 for the i:th layer where class $i$ were present in the labeled image. A simple example with three classes of how this process was done is illustrated in Figure 3.7.

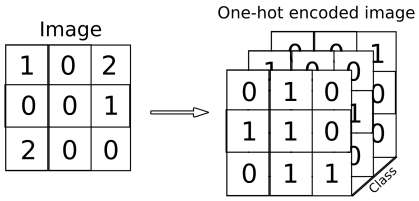



**Figure 3.7:** An example of one-hot encoding of a $3 \times 3$ image with 3 classes

### 3.3.3 Resizing images

Another preprocess that was done before training was to shrink the size of all the images. The size of an input decides the size of all the feature maps which had to be pre-allocated during training. Since the networks were quite complex there was a limitation of how much memory space that could be allocated. The size of the original images in Cityscapes was too large for the networks regarding memory space and was instead resized to $384 \times 256$.

### 3.3.4 Depth predictions from PyD-Net

A limitation in this project was to not build and train an unsupervised depth estimation network. But to stay within the scope to actually fuse two deep neural networks together we still had to get predictions from one. The network chosen to generate depth predictions was PyD-Net with the pre-trained weights obtained by M. Poggi *et al.* [64]. By simply feeding PyD-Net with all the images from the dataset predicted depth estimations were obtained.

### 3.3.5 Normalization of input data

Considering how the networks are updating their weights explained in subsection 2.2.5 we decided to normalize our inputs. Without normalization the values between each feature can end up being too high which will lead to some weight update corrections being either over or under compensated. This can affect a network such that the model will never reach an optima and instead fluctuate or diverge. This problem can be avoided by using normalization [60]. The RGB images from the datasets contains values between [0-255] for each color channel and the depth images predicted by PyD-Net as well. By dividing both the RGB and depth image with 255 the range of the inputs to the network became [0-1] instead.

## 3.4 Training

The same training method were used for every network such that the comparison would be fair. The networks were trained three times separately with the different splits of the Cityscapes dataset.

The loss function used in this project was the Categorical Cross-Entropy, as it is a multi-class loss function that was fit for our project. It was chosen over other multi-class loss functions as it mathematically fulfils the requirement to penalizes confident predictions that was wrong but also predictions which were correct but with high uncertainties.

The used optimizer in the project is SGD. It is a popular optimizer within machine learning and is recommended by Ashia C. Wilson *et al.* and Léon Bottou [77, 8]. The main reason this optimizer was chosen above other popular optimizers like Adam [41] is because SGD is good at generalizing which was more important than fast convergence in this project [63]. We used a momentum 0.9, an initial learning rate 0.01 and a batch size of 6.

After every $\frac{1}{5}th$ epoch we evaluated the loss on the validation set. Even if the weights were updated to descend to the optima it did not always result in lower validation loss after each epoch. One reason for this problem was that the learning rate was too large for some epochs. To fix this problem we decreased the learning rate with $\times 10^{-1}$ if the validation loss had not improved after 2 epochs.

As mentioned, every epoch did not necessarily result in lower loss as the loss usually fluctuates. This means that the weights from the last epoch in the training were not necessarily the best, because of this the weights that had the lowest validation loss was saved. The reason for choosing the validation loss as an indicator and not the training loss, is to reward generalization so the models perform good on data is have not seen before.

The training session of our networks was carefully carried through such that the same result could be achieved and not have a stochastic influence. This was the most important criteria we had because we wanted to make sure that the different networks were trained with the same prerequisites. Firstly, we made sure to use the same hyperparameters, which were the learning rate, momentum and initialization of weights. Secondly, to solve the stochastic influence of SGD but still maintain its attribute we had to have fixed random orders of the data for each epoch.

## 3.5 Analysis

Before we started with experiments, we had to discuss which kind of result we wanted to produce to compare the networks accordingly. Within our scope there were two attributes that we wanted to analyze, the accuracy and the robustness of the networks. How we defined these attributes will be discussed in this section.

### 3.5.1 Accuracy

Accuracy is an important metric and in a lot of papers it is the only measurement of the network's performance. There exist different methods to decide the accuracy. In Chapter 2.7.1 four performance evaluation metrics were presented, *PA*, *MPA*, *MIoU* and *FWIoU* [27].

*PA* differ from the other methods, as it calculates the ratio between number of correct classified pixels to the total number of pixels, while the other three metrics take classes into consideration. As the performance class-wise could contribute to an useful insight of the models total performance, *PA* did not qualify.

To be able to compare the proposed networks performance to other networks in a fair way, the same accuracy metric would need to be used. The three metrics *MPA*, *MIoU* and *FWIoU* is quite similar, though the absolute most common metric to evaluate semantic segmentation models is *MIoU*, both as performance measurement in reports as well as in competitions [27]. For these reasons we decided to use *MIoU* as accuracy metric in the project.

When creating a network to classify pixels on road scenes some classes might be of more importance than other, that's why we also choose to include IoU for the individual classes as a metric. Then it can be observed for which classes the networks performs well on.

### 3.5.2 Robustness

As discussed in section 2.8 researchers had different definitions of what a robust neural network was. The different definitions were *test set robustness* [32], *adversarial robustness* [7, 24] and *noise robustness* [25, 20]. Thus, we had to decide our definition of what a robust network was for this project.

The first definition that was discussed in section 2.8 was how the network perform on an independent but similar dataset. We decided to include this part for our definition of robustness. The kind of experiments from these definitions gave us an approximation of how well the networks performed in normal conditions.

The networks in this project were tested on a test set that was originally from the same dataset as the training set, i.e. Cityscapes. However, the images in this dataset was captured with the same hardware. Therefore, to further test the performance of a network and in a more real-life scenario it was also evaluated on a different dataset from the one it was trained on, in this case KITTI. This resulted in a more generalized and realistic test.

The performance of the networks was evaluated by estimating their accuracy. The metrics which were used, as presented in the accuracy section 3.5.1, were MIoU and IoU.

Resistance against adversarial attacks was the second definition discussed. The task

for our networks was semantic segmentation and how these kinds of attacks could possibly affect them were considered. Digital adversarial attacks were able to create worst-case scenarios for the networks since they could manipulate an input image to achieve a specific output, but to achieve those kinds of attacks were quite advanced. On the other hand, physical adversarial attacks seemed as a bigger threat. However, physical attacks against semantic segmentation has proven to not be so effective [54, 4]. Therefore, we did not include adversarial robustness in our definition of a robust network.

The last robustness definition we have discussed was robustness against noisy data. For some cases it would be reasonable to consider this kind of robustness, for example when identifying patterns in MR images since the images are usually of low quality [20]. However, this type of robustness was outside of the scope for this project and was therefore not considered in our definition of robustness.

MIoU and IoU was decided to be used to evaluate the performance of the networks, though these metrics are rather weak on their own and they are therefore supplemented with other methods to make a more proper comparison, for example the confusion matrix.

Within a decision-making system for autonomous driving some classes will activate more similar actions for the vehicle. For example, the car should not drive on either the class *Vegetation* or *Sidewalk*, so if the network misclassified one as the other it would be considered better than if *Sidewalk* got mixed up with *Road*.

To be able to monitor what the networks predicts for the individual classes we chose to generate the confusion matrix. As described in subsection 2.7.2 the confusion matrix shows the percentage of how many pixels of a class that was correctly classified, miss-classified and for which class they were miss-classified as.

It is hard to use the confusion matrix to give an exact measurement of the network's robustness as it is a combination of multiple result. Though, it is useful tool to get an indication of the robustness, as it gives an insight of how the networks predicts.

Another useful insight of the networks was the consistency of their performance. This was done by estimating the IoU and MIoU for each image and then their mean and standard deviation (std) over all the images were calculated. An accuracy which yield small std indicates that it is consistent to perform as good as the mean value. We decided to include the results from the distributions to support the robustness analysis of a network.

To validate the robustness of the networks for this project we first had to measure the accuracy, both the MIoU and the IoU for the individual classes. This was done by evaluating the networks on a similar independent dataset, i.e. on a separate test set of the datasets KITTI and Cityscapes. Secondly the confusion matrix was generated that described how the networks predicted and lastly, by estimating the distribution of the networks we could analyze how consistent their performance were.

# 4

# Experimental Results

## 4.1 Setup

To carry out the experiments the network was trained on Cityscapes, the performance was evaluated on both Cityscapes and KITTI [15, 28]. The network was trained for 30 epochs with a Categorical Cross-Entropy loss and the SGD optimizer with 0.9 momentum and an initial learning rate 0.01. Decreasing learning rate was used as the network failed to improve. This is more thoroughly discussed in the Chapter 3.4.

The graphic processing unit used for training the networks was a GeForce RTX 2080 from Nvidia [57]. The tools that were used to build and train the models were mostly taken from the Keras Functional API framework [13] and some were custom written as well.

## 4.2 Learning trends

**Table 4.1:** The MIoU over three rounds with shuffled data for the different models. [1] and [2] represent testing on the Cityscapes respectively KITTI dataset. Bolded values indicates the best model for the evaluated dataset row.

| **MIoU** | | Model | | | | | |
|---|---|---|---|---|---|---|---|
| | | SegNet | P-SegNet | DF-SegNet | D-SegNet | BD-SegNet | EF-SegNet |
| Round 1 | Train | 56.3 | **58.6** | 53.8 | 54.4 | 54.6 | 47.2 |
| | Val | 52.7 | **53.6** | 51.3 | 50.9 | 51.4 | 45.6 |
| | Test[1] | 51.6 | **52.6** | 50.6 | 49.3 | 50.2 | 44.9 |
| | Test[2] | 29.2 | **31.2** | 28.8 | 30.7 | 30.6 | 25.0 |
| Round 2 | Train | 52.2 | 52.0 | 52.2 | 49.9 | **53.3** | 47.4 |
| | Val | **50.8** | 50.4 | 50.6 | 48.3 | 49.2 | 45.2 |
| | Test[1] | **46.8** | 46.2 | 46.7 | 44.7 | **46.8** | 41.5 |
| | Test[2] | 29.9 | 31.5 | **31.6** | 28.3 | 28.3 | 21.7 |
| Round 3 | Train | 51.2 | 50.7 | 53.6 | **54.9** | 51.1 | 49.8 |
| | Val | 47.8 | 48.2 | **48.5** | 48.4 | 46.5 | 45.0 |
| | Test[1] | 48.3 | 48.4 | **49.2** | 48.5 | 47.02 | 44.3 |
| | Test[2] | 29.2 | 29.05 | 29.92 | 27.37 | **30.3** | 23.9 |

To evaluate if the training method would yield consistent results the models was trained three times. For each training round the data was randomly rearranged between the categories training, validation and testing, the performance differences for each round are visualized in Table 4.1 and Table 4.2. The tables contains MIoU and loss evaluated on the train, validation and test data from Cityscapes, as well as test data from KITTI, for each training round.

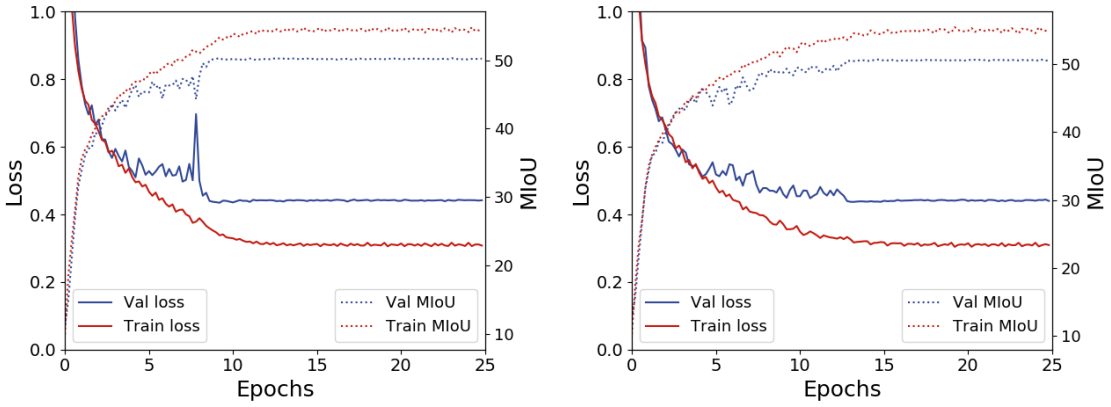
The weights are saved based upon the validation loss, which mean that this is the only metric that is expected to be consistent within our training framework. Therefore it is the only metric that is compared throughout the three rounds. As observed in Table 4.2, the difference of the validation loss between the rounds is insignificant, it can therefore be concluded that the training method is consistent and independent of the order in the dataset.

**Table 4.2:** The Categorical Cross-Entropy loss over three rounds with shuffled data for the different models. [1] and [2] represent testing on the Cityscapes respectively KITTI dataset. Bolded values indicates the best model for the evaluated dataset row.

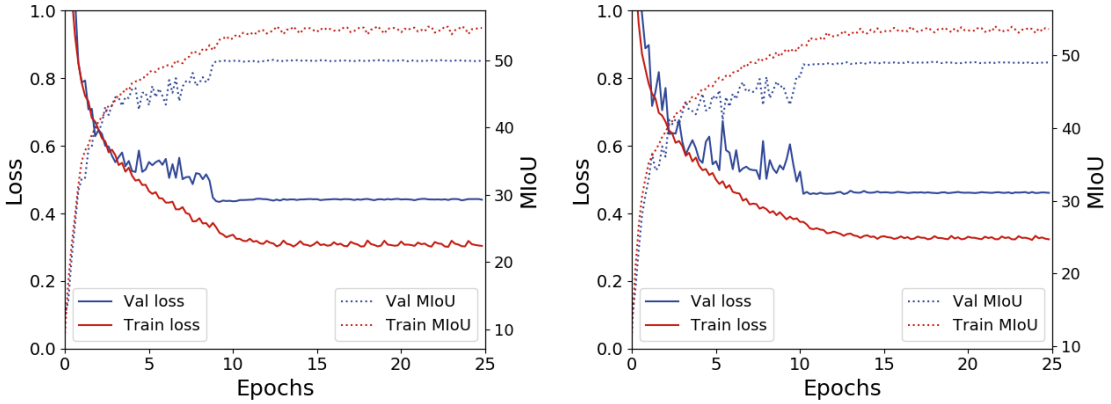
| **LOSS** | | Model | | | | | |
|---|---|---|---|---|---|---|---|
| | | SegNet | P-SegNet | DF-SegNet | D-SegNet | BD-SegNet | EF-SegNet |
| Round 1 | Train | 0.294 | **0.251** | 0.314 | 0.309 | 0.296 | 0.470 |
| | Val | 0.424 | **0.421** | 0.429 | 0.436 | 0.423 | 0.529 |
| | Test[1] | 0.439 | **0.429** | 0.442 | 0.461 | 0.445 | 0.576 |
| | Test[2] | 1.501 | 1.510 | 1.549 | **1.483** | 1.371 | 1.638 |
| Round 2 | Train | 0.374 | 0.367 | 0.368 | 0.392 | **0.318** | 0.476 |
| | Val | **0.421** | 0.428 | 0.423 | 0.464 | 0.428 | 0.535 |
| | Test[1] | **0.503** | 0.514 | 0.504 | 0.538 | 0.515 | 0.634 |
| | Test[2] | 1.446 | 1.376 | **1.358** | 1.536 | 1.587 | 2.019 |
| Round 3 | Train | 0.362 | 0.384 | 0.328 | **0.299** | 0.368 | 0.419 |
| | Val | 0.443 | 0.442 | **0.440** | 0.459 | 0.465 | 0.539 |
| | Test[1] | 0.489 | 0.493 | **0.480** | 0.492 | 0.514 | 0.596 |
| | Test[2] | 1.378 | 1.567 | 1.444 | 1.574 | **1.282** | 1.778 |

An advantage of training the models multiple times is that an average of the models' performances can be estimated. This averaged result will be less biased than a result from only one training round [65]. To take advantage of this property, all the following results will be an average over the performances for each of the three training rounds.

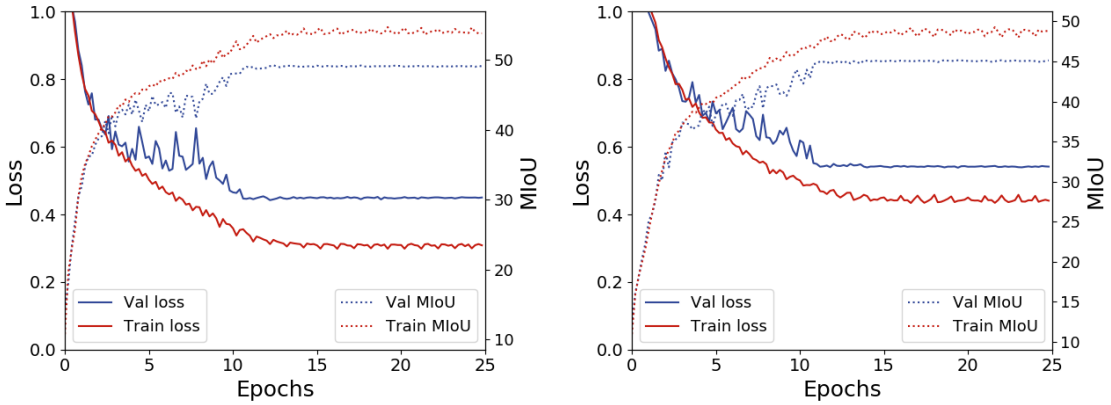
The Figure 4.1 shows the MIoU and loss for the training and validation set during training. It can be seen in the figure that all models and measurements converge, i.e. the derivative level out towards zero, this implies that the networks have found a minimum. The network is trained with a loss function based upon the training loss, and because the training loss converge it implies that the networks manage to resist overtraining towards the training data. The reason the networks manage to converge is mostly due to the regularization i.e. the batch-normalization.




**(a)** SegNet **(b)** P-SegNet




**(c)** DF-SegNet **(d)** D-SegNet

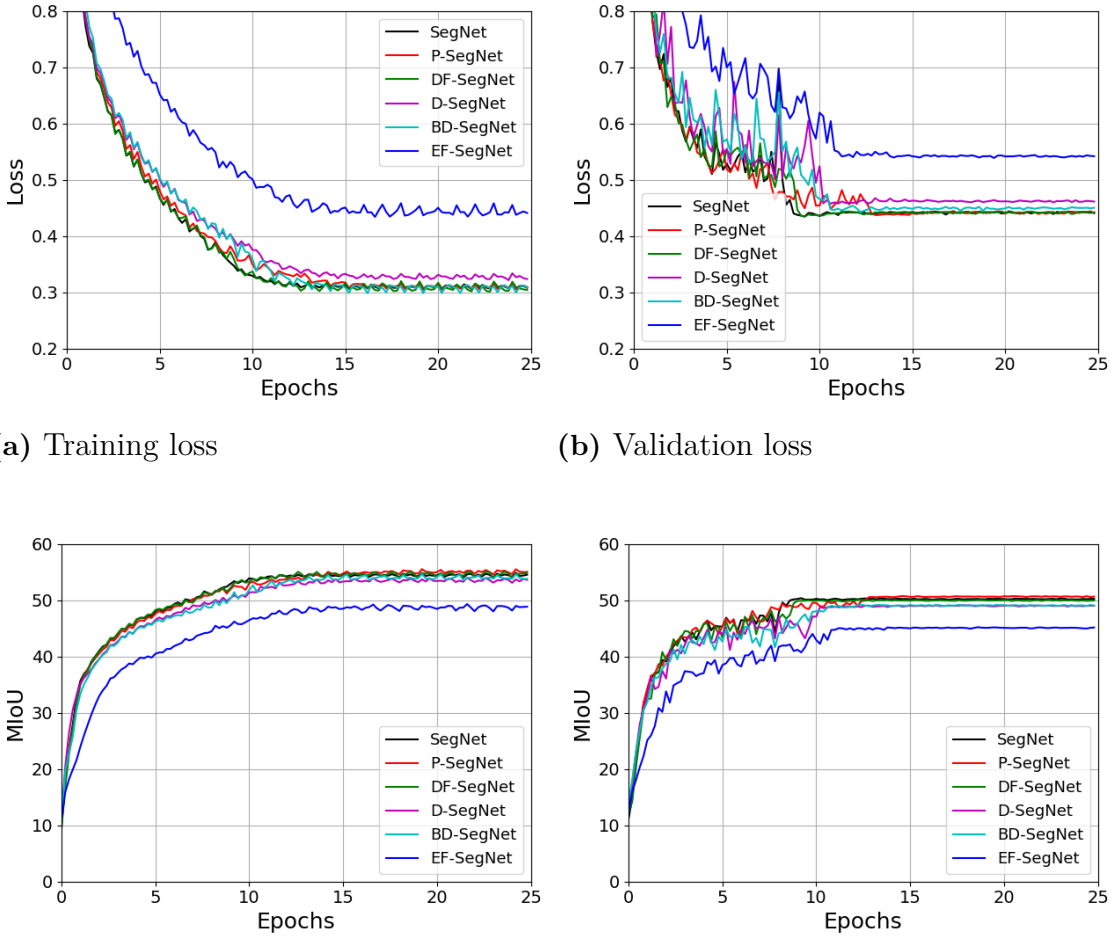



**(e)** BD-SegNet **(f)** EF-SegNet

**Figure 4.1:** Loss and MIoU values for the training and validation set for each model, calculated during the training session.

The figure also shows that the training and validation data, for both MIoU and loss,

starts to diverge from each other after some epochs, leading the training and validation metrics to converge to different values. This is another type of overfitting than the one mentioned in the paragraph above and is harder to avoid. This overfitting occurs as the models learns features that is unique to the training set, and as these features do not exist in the validation set the model will perform worse on this set. EF-SegNet, is the model that have the lowest difference between the validation and training metrics, meaning it overfit least in this type of overfitting.




**(a)** Training loss

**(b)** Validation loss

**(c)** Training MIoU

**(d)** Validation MIoU

**Figure 4.2:** Loss and MIoU values for the training and validation set for all the models, calculated during the training session.

As mini-batches are used in this project, one update of the weights, i.e. one step, is with respect to a specific mini-batch. The step will be taken in a sub-optimal direction and as the mini-batch does not represent the whole dataset perfect, this will occasionally result in an update that would increase the validation loss. After one fifth of an epoch the validation metrics are estimated and can possibly yield a larger validation loss due to the weights being updated too bias to the training set. With a large learning rate this can lead to large fluctuations of these validation

metrics, this occurrence can be seen for the models in the beginning of the training in Figure 4.1. As the learning rate decreases throughout the training, the fluctuation decreases as well.

In Figure 4.2 the training and validation metrics are separated such that they could be compared between the different models. The model that stand out mostly in this figure is EF-SegNet. It has the highest loss and lowest MIoU on both the training and validation data. The rate of the convergence is very similar for the different models, they all converge around epoch fifteen, judging by the training loss. There are a lot of factors that dictates the convergence speed, for example, the learning rate, loss function, dataset, complexity of task and architecture. The difference between the models is their architecture, which also is not that dissimilar. Therefore, it is not unreasonable that they converge around the same time.

The validation metrics for D-SegNet and BD-SegNet fluctuates more in the beginning of the training than they do for the other models. The behaviour is not surprising as these two models have more parameters and the network therefore have more parameters to tune. With a high learning rate this can lead to large changes in the weights and therefore fluctuations.

## 4.3 Accuracy

The Table 4.3 shows the Mean Intersection over Union (MIoU) in Equation 2.18 and the class-wise Intersection over Union (IoU) for all the models, generated on the test set from Cityscapes.

**Table 4.3:** Mean Intersection over Union (MIoU) in Equation 2.18 and class-wise Intersection over Union (IoU) comparison between the six models. The table was generated using images from the Cityscapes test set.

| | | **Model** | | | | | |
|---|---|---|---|---|---|---|---|
| | | SegNet | P-SegNet | DF-SegNet | D-SegNet | BD-SegNet | EF-SegNet |
| | **MIoU** | 48.57 | **48.61** | 48.48 | 47.07 | 47.84 | 43.24 |
| **Class IoU** | Void | 54.55 | **54.67** | 54.63 | 52.79 | 54.13 | 48.73 |
| | Road | **90.32** | 90.13 | 90.14 | 89.3 | 89.53 | 85.67 |
| | Sidewalk | 40.79 | **41.03** | 40.98 | 37.15 | 37.75 | 30.82 |
| | Building | 66.81 | **67.23** | 66.99 | 65.35 | 67.11 | 62.23 |
| | Pole | 0.01 | **0.78** | 0.01 | 0.04 | 0.16 | 0.0 |
| | Vegetation | 68.78 | 69.02 | 68.79 | 68.42 | **69.08** | 61.66 |
| | Terrain | 14.86 | 14.59 | **15.21** | 14.25 | 13.66 | 8.40 |
| | Sky | 73.05 | **73.68** | 73.53 | 72.82 | 72.09 | 68.61 |
| | Person | 3.42 | 4.39 | 3.12 | 3.19 | **6.45** | 0.01 |
| | Car | **59.74** | 59.52 | 59.44 | 55.89 | 57.36 | 49.11 |

P-SegNet performs best regarding MIoU and IoU for half of the classes. Though, SegNet and DF-SegNet have only slightly lower MIoU which mean that the difference is to insignificant to declare that P-SegNet is best. The result shows clearly that

EF-SegNet, performs considerably worse than the other models. It has the lowest MIoU, but also the lowest IoU for all the classes. Therefore, it can be concluded that EF-SegNet is worse than the other models.

The classes *Road*, *Sky*, *Vegetation* and *Building* have the highest accuracies while *Pole* and *Person* have the lowest, this is true for all models. This means that the models are certain with their predictions for some classes and uncertain for some, the reason why is discussed further in chapter 5.

The MIoU and IoU evaluated from KITTI can be seen in Table 4.4. Similar conclusions as the previous results in Table 4.3 regarding the performance of the models can be drawn. P-SegNet still performs marginally better, but for this case all the other models, except for EF-SegNet, have similar MIoU. The classes *Terrain* and *Sky* have similar IoU in both of the results for Cityscapes and KITTI, while the classes *Sidewalk*, *Building* and *Void* have considerably lower IoU on KITTI.

**Table 4.4:** MIoU and class-wise IoU comparison between the six models. The table was generated using images from the KITTI test set. *The numbers represent the models in order: SegNet, P-SegNet, DF-SegNet, D-SegNet, BD-SegNet and EF-Segnet.*

| | | **Model** | | | | | |
|---|---|---|---|---|---|---|---|
| | | SegNet | P-SegNet | DF-SegNet | D-SegNet | BD-SegNet | EF-SegNet |
| | **MIoU** | 29.4 | **30.6** | 30.1 | 28.8 | 29.7 | 23.5 |
| **Class IoU** | Void | 12.55 | **12.94** | 12.52 | 10.69 | 12.09 | 7.83 |
| | Road | 60.95 | 60.99 | 61.32 | 60.07 | **62.64** | 48.21 |
| | Sidewalk | **8.35** | 8.07 | 8.16 | 5.70 | 5.66 | 4.12 |
| | Building | 15.71 | 16.98 | 17.56 | **21.04** | 18.4 | 15.6 |
| | Pole | 0.00 | **0.60** | 0.02 | 0.22 | 0.22 | 0.0 |
| | Vegetation | 56.2 | 56.85 | 58.78 | 54.63 | **60.89** | 46.86 |
| | Terrain | 17.63 | 20.0 | **20.4** | 13.64 | 17.37 | 3.38 |
| | Sky | 73.45 | **76.3** | 74.75 | 73.6 | 72.55 | 66.4 |
| | Person | 0.44 | 0.63 | 0.76 | 0.38 | **0.65** | 0.0 |
| | Car | 40.14 | **44.74** | 38.99 | 41.37 | 44.39 | 25.14 |

These results can also be compared to evaluate if the networks are robust according to the definition made in this report. One of the criterions to determine if a network is robust is the general performance on data it have not seen before. Looking at the results it can be concluded that the models can be defined as robust for that criterion for the classes *Terrain* and *Sky*.

## 4.4 Predictions

In Figure 4.3 and Figure 4.4 the models' predictions on four images from Cityscapes test set respectively on four images from KITTI is shown. This is visualized to give an broader understanding of the models performance. The weights from round 1 was used to generate the predictions since it was the training round that gave the

overall highest MIoU. More predictions of images from the Cityscapes respectively KITTI can be found in Appendix B.

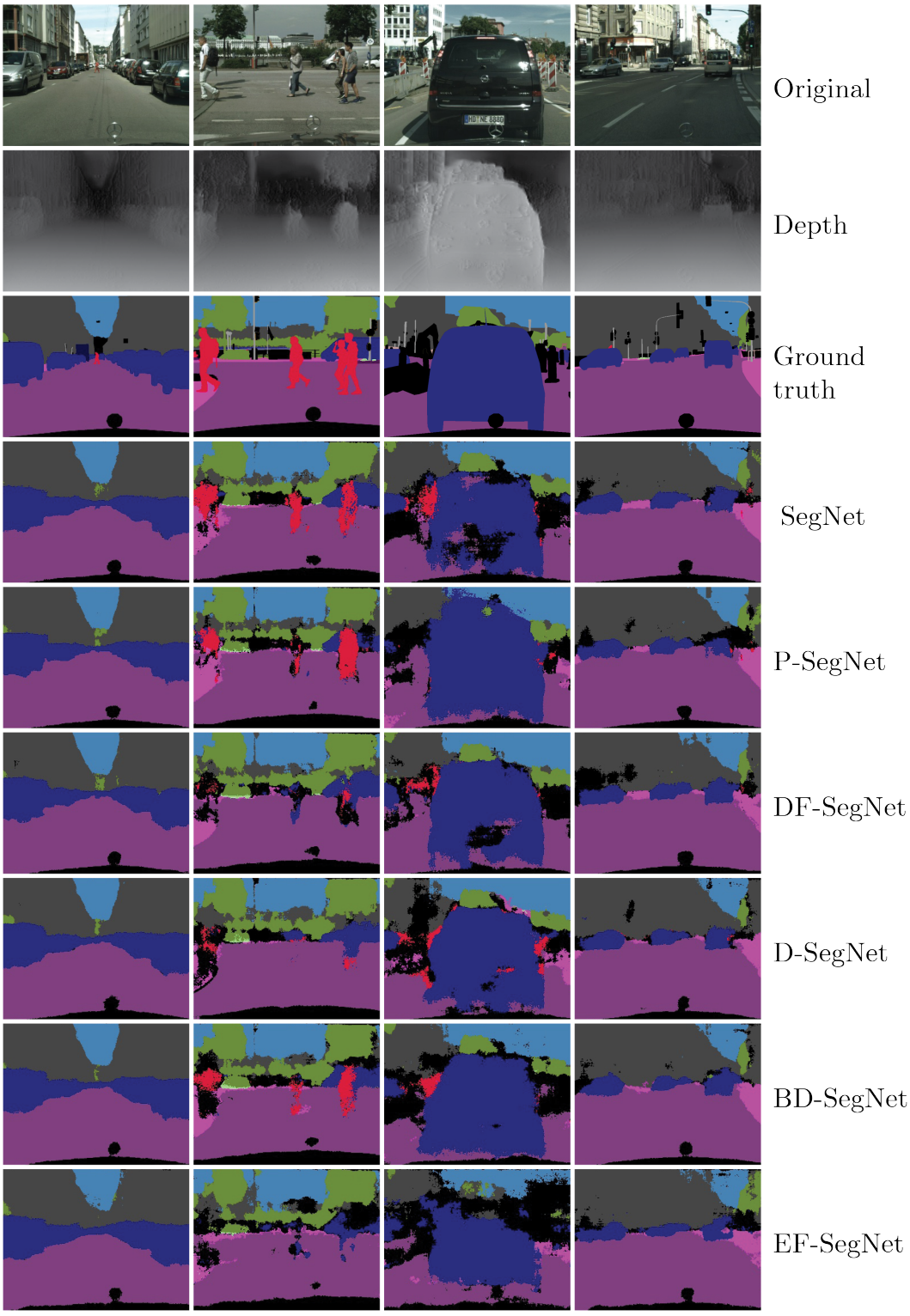



**Figure 4.3:** Predictions of four images from Cityscapes. From top to bottom: Original image, corresponding depth map, pixel-wise classified ground truth, pixel-wise predictions from the six different models.

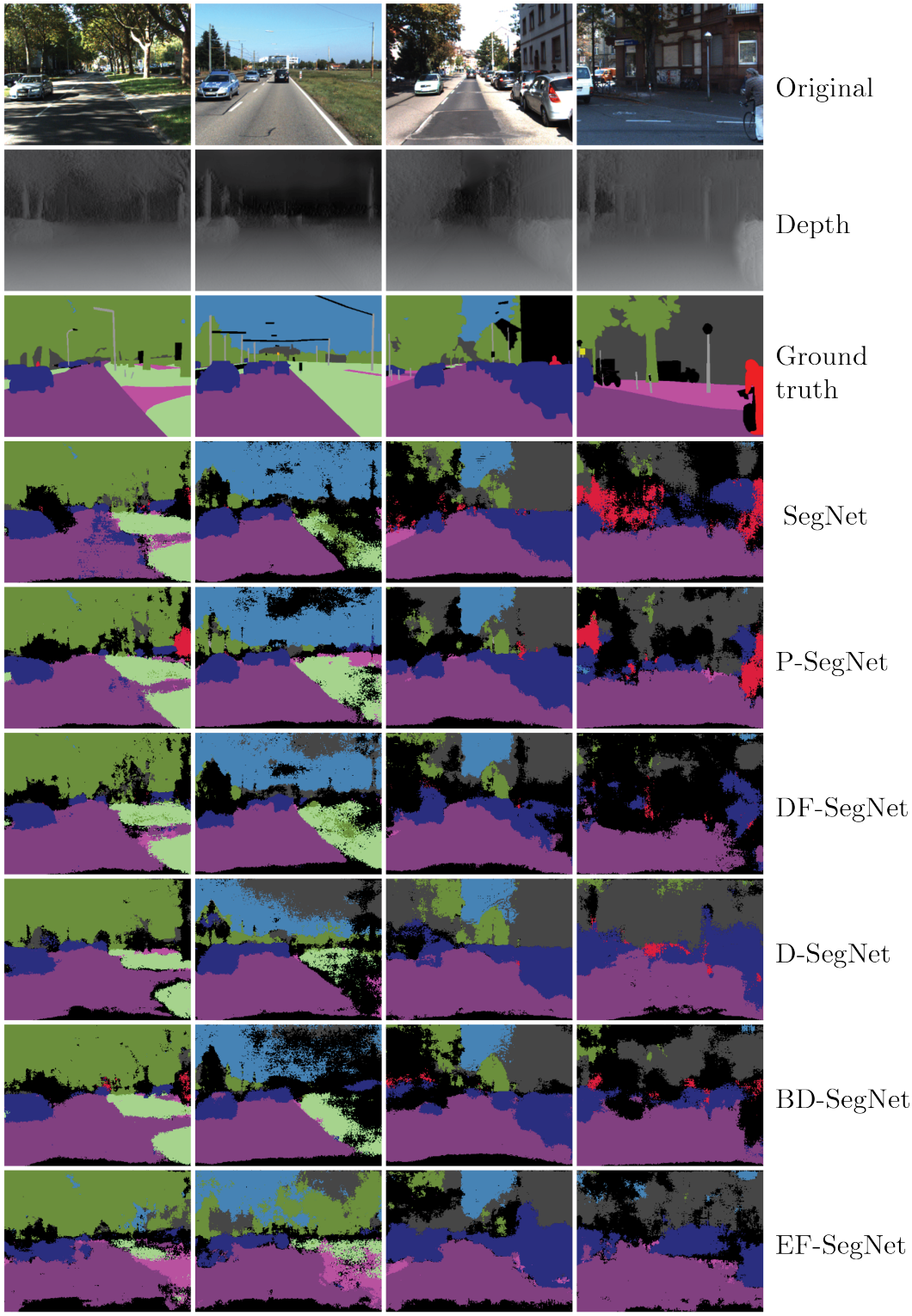



**Figure 4.4:** Predictions of four images from KITTI. From top to bottom: Original image, corresponding depth map, pixel-wise classified ground truth, pixel-wise predictions from the six different models.

In Figure 4.3 the models managed to predict the classes *Road*, *Vegetation*, *Building* and *Sky*, all the classes that had a high IoU. The prediction on the classes *Car* and

*Person* varies between the models. P-SegNet, have the most accurate prediction while EF-SegNet, do not manage to predict the class *Person* at all and very poorly for *Car*.

The predictions on the image from KITTI is not as good as the predictions on the image from Cityscapes. It is to be expected as the models was trained on Cityscapes and the datasets are slightly different. The images of the datasets was captured using different hardware, which effects the hue, saturation and exposure of the image. The images also have different dimensions in $Width \times Heigth$, $2048 \times 1024$ in Cityscapes and $1242 \times 375$ in KITTI. Though, the basic features for the classes remains the same between the datasets which explains why the networks can predict some of the classes in the image.

## 4.5 Confusion matrix

The confusion matrices are presented in this section, and as described in subsection 2.7.2, they are remade to show proportions for easier analysis. The columns shows which class the network have predicted on. As example in Table 4.5, the value for index **Confusion**(*Sidewalk*, *Building*) is 0.4 which mean that SegNet predicted *Sidewalk* when the ground truth was *Building* 0.4 % of the occasions when *Building* was present. In other words, the column represents how certain the network is with its predictions. If the columns *Building* and *Terrain* are compared it can be noted that the network predicts *Building* quite frequent for different classes, but it almost only predicts *Terrain* when the ground truth is *Terrain* which mean that the network is certain when it predicts *Terrain*.

**Table 4.5:** Confusion matrix for SegNet trained on Cityscapes. The diagonal shows the proportional correct classified pixels and the rows the proportional of the misclassified pixels for each class. *Note that this is not IoU accuracy, the relationship is explained in subsection 2.7.2*

| | | **Predicted class** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Actual class** | | Void | Road | Sidewalk | Building | Pole | Vegetation | Terrain | Sky | Person | Car |
| | Void | **63.8** | 5.1 | 4.2 | 16.8 | 0.0 | 4.0 | 0.5 | 0.5 | 0.6 | 4.5 |
| | Road | 1.7 | **95.4** | 2.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.0 | 0.1 | 0.7 |
| | Sidewalk | 9.2 | 16.6 | **70.0** | 1.7 | 0.0 | 0.1 | 0.9 | 0.0 | 0.2 | 1.3 |
| | Building | 3.9 | 0.0 | 0.4 | **90.8** | 0.0 | 3.0 | 0.0 | 0.6 | 0.2 | 1.1 |
| | Pole | 20.6 | 0.6 | 6.2 | 47.6 | **0.0** | 17.6 | 0.6 | 1.8 | 1.7 | 3.1 |
| | Vegetation | 2.5 | 0.1 | 0.2 | 8.0 | 0.0 | **87.6** | 0.5 | 0.5 | 0.0 | 0.6 |
| | Terrain | 16.2 | 5.4 | 11.4 | 0.8 | 0.0 | 10.4 | **54.3** | 0.0 | 0.2 | 1.3 |
| | Sky | 0.7 | 0.0 | 0.0 | 5.6 | 0.0 | 1.6 | 0.0 | **92.1** | 0.0 | 0.0 |
| | Person | 40.6 | 3.7 | 3.2 | 17.8 | 0.0 | 3.0 | 0.1 | 0.0 | **15.8** | 15.8 |
| | Car | 4.5 | 2.4 | 0.3 | 3.1 | 0.0 | 0.8 | 0.1 | 0.0 | 0.2 | **88.7** |

The proportional confusion matrix for SegNet and P-SegNet is presented in Table 4.5 respectively Table 4.6, the remaining classes confusion matrices can be found in Appendix A. The classes with a high IoU in Table 4.3 also have a high score in

the confusion matrices. This is to be expected since the confusion matrix is used to calculate the IoU. What can be seen in the confusion matrix, that can not be seen by looking at the IoU, is both the correctly classified and miss-classified predictions compared to the ground truth.

**Table 4.6:** Confusion matrix for P-SegNet trained on Cityscapes. The diagonal shows the proportional correct classified pixels and the rows the proportional of the misclassified pixels for each class. *Note that this is not IoU accuracy, the relationship is explained in subsection 2.7.2*

| | | **Predicted class** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Actual class** | | Void | Road | Sidewalk | Building | Pole | Vegetation | Terrain | Sky | Person | Car |
| | Void | **64.1** | 4.9 | 4.2 | 16.3 | 0.0 | 4.2 | 0.6 | 0.6 | 0.7 | 4.5 |
| | Road | 1.9 | **95.3** | 1.9 | 0.0 | 0.0 | 0.0 | 0.1 | 0.0 | 0.0 | 0.7 |
| | Sidewalk | 9.6 | 16.9 | **69.4** | 1.5 | 0.0 | 0.1 | 1.1 | 0.0 | 0.2 | 1.2 |
| | Building | 3.8 | 0.0 | 0.4 | **90.5** | 0.0 | 3.2 | 0.0 | 0.6 | 0.2 | 1.1 |
| | Pole | 21.8 | 0.6 | 6.0 | 44.8 | **1.2** | 18.0 | 0.8 | 1.8 | 2.0 | 3.1 |
| | Vegetation | 2.6 | 0.1 | 0.2 | 7.3 | 0.0 | **88.0** | 0.6 | 0.6 | 0.1 | 0.6 |
| | Terrain | 15.3 | 5.7 | 10.2 | 0.9 | 0.0 | 10.7 | **55.5** | 0.0 | 0.1 | 1.6 |
| | Sky | 0.6 | 0.0 | 0.0 | 4.5 | 0.0 | 1.6 | 0.0 | **93.3** | 0.0 | 0.1 |
| | Person | 39.1 | 3.6 | 2.6 | 16.3 | 0.1 | 2.8 | 0.1 | 0.0 | **18.0** | 17.3 |
| | Car | 4.4 | 2.6 | 0.3 | 2.9 | 0.0 | 0.7 | 0.1 | 0.0 | 0.3 | **88.7** |

The biggest difference between the confusion matrices for SegNet and P-SegNet is in the class *Pole*. SegNet never predicts the class at all while P-SegNet predict the class correct 1.2 % and miss-predict the class *Person* as *Pole* 0.1 %. Apart from this class there is little difference between the confusion matrices.

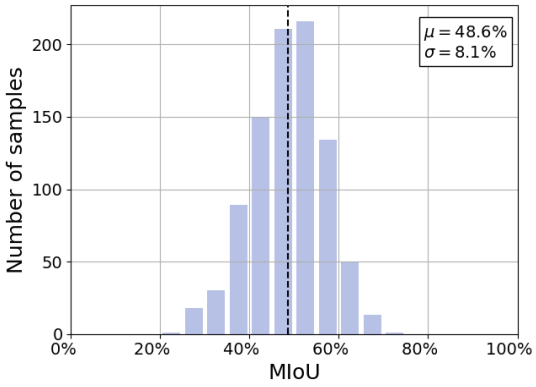
Apart from discussing the difference between the matrices for the two models the classes can be discussed. Some classes give interesting insight, to discuss these classes we use the confusion matrix from P-SegNet as a reference, as the matrices are so similar is should not make an impact. As said before, the proportional of correct classified pixels in P-SegNet are very low for the class *Pole*, 1.2 %. When P-SegNet predicts a pixel which in reality have the class *Pole*, it 44.8 % of the time predict is as the class *Building* and 21.1 % as the class *Void*. What is interesting is that a pixel representing the class *Building* or *Void* never gets miss-classified as *Pole*.

A pixel representing either *Sky* or *Road* is most of the times correct classified. What sets them apart is the percentage of times that other classes is predicted as them. For the class *Sky* this barely happens, for *Road* however this happens relatively often for pixels representing the class *Sidewalk*, *Void* and *Terrain*. The class *Void* is a more extreme case of *Road*, the predictions of pixels representing *Void* is fairly good, however pixels representing other classes gets predicted as *Void* a lot.

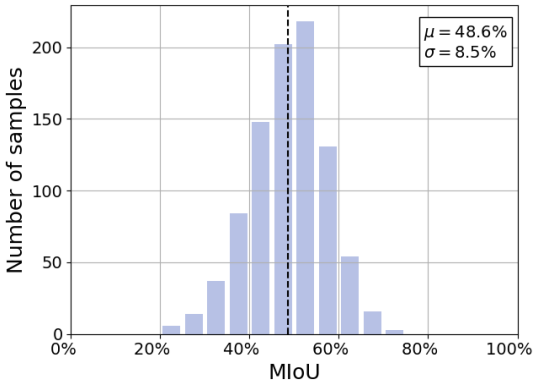
## 4.6 Distribution

The distribution of the MIoU and the IoU for certain classes on P-SegNet and SegNet will be presented in this chapter, the remaining classes, as well as MIoU for all

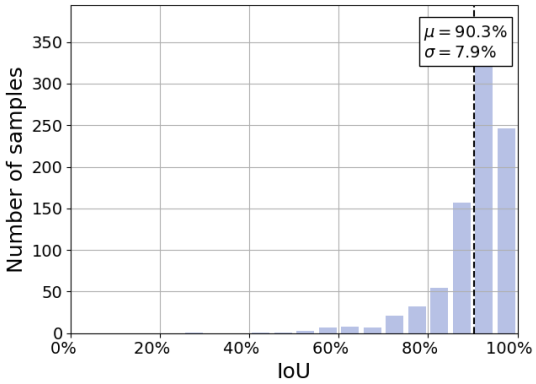
models, can be found in Appendix C. The process of generating these distributions are explained in subsection 3.5.2.



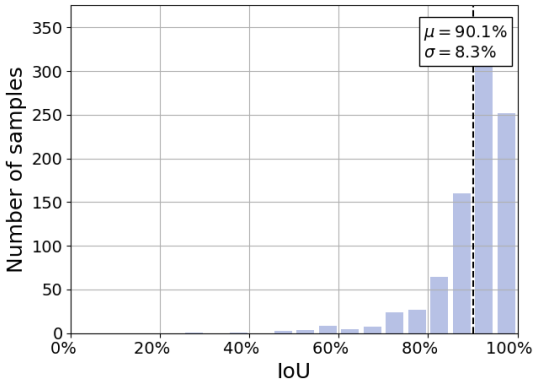

**(a)** SegNet




**(b)** P-SegNet




**(c)** Class *Road* on SegNet

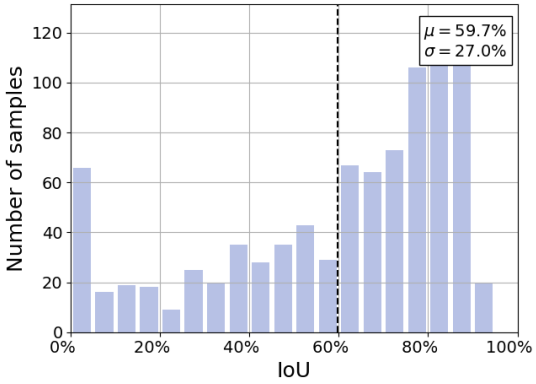



**(d)** Class *Road* on P-SegNet

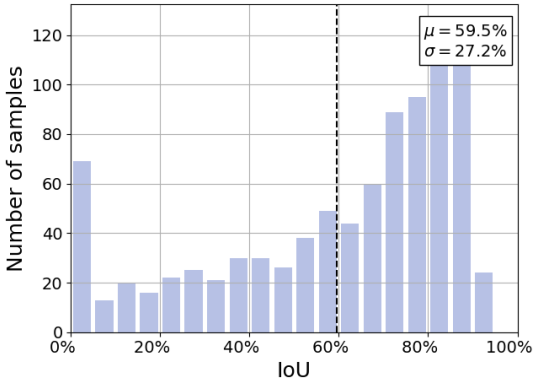
The MIoU distributions can be seen in Figure 4.5a and 4.5b. The distribution is very similar between the two models, though SegNet have slightly lower standard deviation. This means SegNet can be trusted to give more consistent predictions that is closer to the average MIoU, making it more robust.

The class-wise distributions are given in Figure 4.5c-j. The classes that are presented is *Road*, *Car*, *Pole* and *Terrain*, these classes cover together all unique results which is the reason they were chosen. The result between the classes are, as expected considering the distribution of MIoU, roughly the same. The largest difference can be seen in the class *Pole* were PydSegNet have somewhat higher average IoU resulting in an also increasing standard deviation. This was anticipated as the confusion matrix for SegNet, in Table 4.5, showed zero prediction of the class *Pole*, while PydSegNet, in Table 4.6, had some few percentage correct classified pixels.

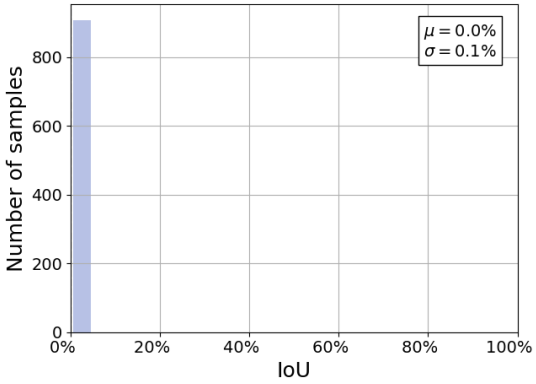
*Road* is the only class that have high average IoU and low standard deviation, making it the only robust class regarding consistency. The *Car* class have decent



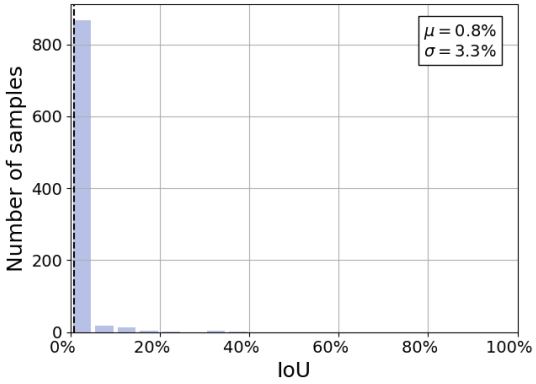

**(e)** Class *Car* on SegNet



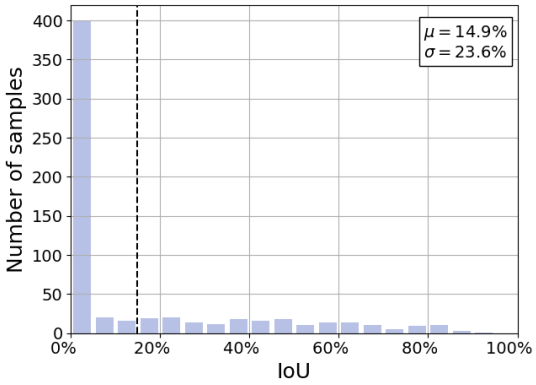

**(f)** Class *Car* on P-SegNet



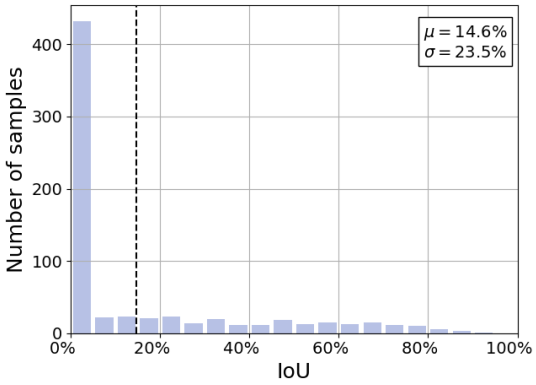

**(g)** Class *Pole* on SegNet




**(h)** Class *Pole* on P-SegNet




**(i)** Class *Terrain* on SegNet




**(j)** Class *Terrain* on P-SegNet

**Figure 4.5:** Distribution of MioU and IoU for the classes *Road*, *Car*, *Pole* and *Terrain* on SegNet and P-SegNet, generated by calculation the accuracy for every image in Cityscapes test set. The dashed line represents the averaged MIoU which is also presented in the top-right box together with the standard deviation.

high average IoU but a very large standard deviation. The class have 60+ predictions of images with an IoU of 0-5 % resulting in unreliable predictions and therefor low robustness. *Terrain* almost have as high standard deviation as *Car*, however the class IoU mode, i.e. the most repeated IoU, is clearly 0-5 %

# 5

# Discussion

Based on the results in Table 4.3 and Table 4.4 it seems like the models, except EF-SegNet, which had depth predictions from PyD-Net as input performed similar regarding MIoU. Among all the models it seems like P-SegNet scores the highest MIoU value but just with a small margin. Some of the reasons why the model performs similar can be explained. Firstly, the same data have been exposed for the models. Secondly, the models have been trained through the same prerequisites regarding the training method. Lastly, the architectures are quite similar to each other since they all are designed as SegNet and process the inputs with convolutional layers. But there can be additional factors that affect why the networks performs similar.

Between the proposed networks and SegNet there were no noticeable differences in the results, and the reason why is hard to pinpoint. Both PyD-Net and SegNet are CNNs with an encoder and decoder, which mean that both networks extract features such as edges in the encoder to determine either depth or classes. A hypotheses is, that the same information is extracted from both a RGB and depth image, when processing through convolutional layers. This means that the additional depth image would not supply the network with any new information, thus the lack of differences between SegNet and the proposed fusion networks. The proposed MTL model by A. Kendall *et al.* were designed to solve *semantic segmentation*, *instance segmentation* and *depth estimation* and only consists of one encoder followed by three separate decoders for each task [40]. Their model is an evidence that the same features extracted from an image can provide information to both distinguish classes and depth.

A better approach for this project could have been to focus on different methods to process the information in a depth image rather than test different CNN architectures. As an example the depth could have been processed through a method called depth-aware convolutional layer proposed by W. Wang and U. Neumann [76]. Compared to a convolutional layer it takes two inputs instead, the feature map and the depth image where the depth information monitors the scale of the feature value based on the depth difference between neighbouring pixels. Another example could be to design a MTL network similar to the one proposed by A. Kendall *et al.* [40] but instead use predicted depth images as the target instead of annotated depth data as they did. We highly recommend that these methods should be carried through for future work that will have the same aim as this project.

Although it is a small margin it is still interesting to discuss why P-SegNet performs better than the rest of the models. P-SegNet, DF-SegNet and SegNet have the same architecture and the only difference between them is how the depth image have been fused or for SegNet not at all. These are also the top three performing models out of the six. In P-SegNet the depth image is concatenated with the RGB image before it is fed to the layers compared to DF-SegNet where the depth image is concatenated with the feature maps after the last upsampling layer. This means that the depth image is only processed through two convolutional layers in DF-SegNet and might be the reason why it performs worse than P-SegNet. D-SegNet and BD-SegNet use another approach to process the depth image which is with a second encoder-decoder. They performed a bit worse compared to the other top-three and among themselves D-SegNet was the worst. But the network that differentiated among them all was EF-SegNet which performed the worst. EF-SegNet do not have the same amount of upsampling layers as max-pooling layers which makes upsampling with saved max-pooling indices complicated. A lazy approach that was carried through for EF-SegNet was to only use the max-pooling indices from the RGB encoder to determine the upsampling indices. This is probably one of the reasons why EF-SegNet performs the worst.

In the result it was concluded that the classes *Pole*, *Terrain* and *Person* is poorly classified independent on model or training round, this could for example be seen in Table 4.3. These three classes have one important thing in common, they are all very badly represented in Cityscapes. As seen in Table 3.1 these classes have representation around 1 % which is the lowest of all classes. As was seen in the confusion matrices in Table 4.5 and 4.6, *Poles* is often predicted as *Building*, but pixels representing *Buildings* is never predicted as *Pole*. One reason could be that compared to *Poles*, *Buildings* is a lot more represented in the dataset. The networks have therefore trained more to recognize buildings giving the networks a better chance to learn unique features of buildings. The features of a *Pole* could, in addition to this, be quite similar to some of the features in a *Building*, but as buildings are more complex in their structure, they could have additional features that poles do not have.

*Terrain* is a class that would sometimes be predicted as *Void*, *Road*, *Sidewalk* or *Vegetation*. If we disregard the *Void* class for now, a conclusion of this behavior can be drawn. The features of *Road*, *Sidewalk* and *Terrain* can be quite similar as all these classes represent different categories of the ground. When considering the miss-classification as *Vegetation*

# 6

# Conclusion

The aim throughout this project have been to study if a fusion of two deep neural networks could affect their performances. In particular, if depth predictions from PyD-Net would provide additional information that could affect the performance of pixel-wise classification through SegNet. Five architectures was proposed to process the depth image with different approaches. The results from the project concluded that, with the method used, the depth information neither increased nor decreased the performance of the networks. Although, there is still a possibility that a fusion will affect the networks with another methodology and is recommended to investigate for future work. The result from the project can therefore neither prove or disprove, that a fusion of two deep neural networks can affect the networks performance.

A secondary aim was to investigate and define the robustness of deep neural networks. To be able to conclude that the networks are robust they had to have similar MIoU on Cityscapes as on KITTI, a low standard deviation and perform arguably good on their confusion matrix. Since the networks MIoU differs on the datasets evaluated on, they cannot be declared as robust. The MIoU however have relatively low standard deviation, this makes us certain that the networks will give consistent predictions with a performance around the MIoU for every image.

By evaluating the robustness regarding the classes with the same definition, it can be concluded that *Sky* and *Terrain* have both similar IoU for Cityscapes and KITTI, and that if these two classes are predicted, it is high probability that the prediction is correct. This check two of the boxes for robustness, however their standard deviation is above 20 % and cannot be considered low. *Road* have a standard deviation below 10% which shows that the predictions are consistent, making the class robust in that sense.

Overall, this project has concluded that the performance of a semantic segmentation network can not be increased by directly fusing it with a depth network producing inferior depth images.

# Bibliography


[1] H. Alhaija, S. Mustikovela, L. Mescheder, A. Geiger, and C. Rother. Augmented reality meets computer vision: Efficient data generation for urban driving scenes. *International Journal of Computer Vision (IJCV)*, 2018.

[2] H. Alhaija, S. Mustikovela, L. Mescheder, A. Geiger, and C. Rother. Augmented reality meets computer vision: Efficient data generation for urban driving scenes, 2018.

[3] A. Arnab, O. Miksik, and P. H. Torr. On the robustness of semantic segmentation models to adversarial attacks, 2018.

[4] A. Arnab, O. Miksik, and P. H. Torr. On the robustness of semantic segmentation models to adversarial attacks, 2018.

[5] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation, 2017.

[6] R. Barth, J. IJsselmuiden, J. Hemming, and E. V. Henten. Data synthesis methods for semantic segmentation in agriculture: A capsicum annuum dataset. *Computers and Electronics in Agriculture*, 144, 2018.

[7] A. Boloor, X. He, C. D. Gill, Y. Vorobeychik, and X. Zhang. Simple physical adversarial examples against end-to-end autonomous driving models. *CoRR*, 2019.

[8] L. Bottou. Stochastic gradient descent tricks, 2012.

[9] N. Carlini and D. A. Wagner. Audio adversarial examples: Targeted attacks on speech-to-text, 2018.

[10] K. Chellapilla, S. Puri, and P. Simard. High performance convolutional neural networks for document processing. *Tenth International Workshop on Frontiers in Handwriting Recognition*, 2006.

[11] L. Chen, G. Papandreou, F. Schroff, and H. Adam. Rethinking atrous convolution for semantic image segmentation. *CoRR*, 2017.

[12] L. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. *CoRR*, 2018.

[13] F. Chollet et al. Keras, 2015.

[14] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber. Deep big simple neural nets excel on handwritten digit recognition. *CoRR*, 2010.

[15] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. *CoRR*, 2016.

[16] G. Csurka, D. Larlus, and F. Perronnin. What is a good evaluation measure for semantic segmentation?, 2013.

[17] CVPR. Conference on computer vision and pattern recognition, 2017.

[18] A. Deshpande. A beginner's guide to understanding convolutional neural networks, 2016.

[19] T. Dettmers. Deep learning in a nutshell: History and training, 2015.

[20] M.-R. D.Unay, A.Ekin and A.Ercil. Robustness of local binary patterns in brain mr image analysis, 2007.

[21] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. *CoRR*, 2014.

[22] M. Everingham, A. S. Eslami, L. Van-Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective, 2015.

[23] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111, 2015.

[24] I. Evtimov, K. Eykholt, E. Fernandes, T. Kohno, B. Li, A. Prakash, A. Rahmati, and D. Song. Robust physical-world attacks on machine learning models. *CoRR*, 2017.

[25] A. Fawzi, S.-M. Moosavi-Dezfooli, and P. Frossard, 2016.

[26] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biol. Cybernetics 36*, 1980.

[27] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, and J. G. Rodríguez. A review on deep learning techniques applied to semantic segmentation. *CoRR*, 2017.

[28] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[29] C. Godard, O. M. Aodha, and G. J. Brostow. Unsupervised monocular depth estimation with left-right consistency. *CoRR*, 2016.

[30] C. Godard, O. M. Aodha, and G. J. Brostow. Unsupervised monocular depth estimation with left-right consistency, 2016.

[31] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples, 2015.

[32] S. Hashem. Optimal linear combinations of neural networks, 1997.

[33] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, 2015.

[34] M. Himmelsbach, T. Lüttel, H.-J. Wünsche, and A. Müller. Lidar-based 3d object perception, 2008.

[35] D. H. Hubel and T. N. Wiesel. Receptive fields of single neurones in the cat's striate cortex., 1959.

[36] N. Hubens. Build a simple image retrieval system with an autoencoder, 2018.

[37] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[38] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, 2015.

[39] A. Kendall, V. Badrinarayanan, and R. Cipolla. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding, 2016.

[40] A. Kendall, Y. Gal, and R. Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. *CoRR*, 2017.

[41] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *arXiv e-prints*, 2014.

[42] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks, 2012.

[43] A. Kurenkow. A 'brief' history of neural nets and deep learning, 2015.

[44] L. Ladický, J. Shi, and M. Pollefeys. Pulling things out of perspective, 2014.

[45] Y. LeCun and C. Cortes. MNIST handwritten digit database, 2010.

[46] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning apllied to document recognition. *Proceedings of the IEEE*, 1998.

[47] G. Leâıtre and E. W. Yong. Evaluation measures for segmentation, 2015.

[48] W. Li, C. He, J. Fang, J. Zheng, H. Fu, and L. Yu. Semantic segmentation-based building footprint extraction using very high-resolution satellite images and multi-source gis data. *Remote Sensing*, 11, 2019.

[49] Y. Liao, S. Kodagoda, Y. Wang, L. Shi, and Y. Liu. Understand scene categories by objects: A semantic regularized scene classifier using convolutional neural networks. *CoRR*, 2015.

[50] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[51] L. Lu, Y. Shin, Y. Su, and G. E. Karniadakis. Dying relu and initialization: Theory and numerical examples, 2019.

[52] M. Menze, C. Heipke, and A. Geiger. Joint 3d estimation of vehicles and scene flow. In *ISPRS Workshop on Image Sequence Analysis (ISA)*, 2015.

[53] M. Menze, C. Heipke, and A. Geiger. Object scene flow. *ISPRS Journal of Photogrammetry and Remote Sensing (JPRS)*, 2018.

[54] J. H. Metzen, M. C. Kumar, T. Brox, and V. Fischer. Universal adversarial perturbations against semantic image segmentation, 2017.

[55] A. Mądry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks, 2017.

[56] M. A. Nielsen. Neural networks and deep learning, 2015.

[57] Nvidia. Geforce rtx 2080 user guide, 2018.

[58] K.-S. Oh and K. Jung. Gpu implementation of neural networks. *Pattern Recognition*, 37, 2004.

[59] A. E. Orhan. Skip connections as effective symmetry-breaking. *CoRR*, 2017.

[60] K. K. Pal and K. S. Sudeep. Preprocessing for image classification by convolutional neural networks. *2016 IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*, 2016.

[61] R. Pascanu, T. Mikolov, and Y. Bengio. Understanding the exploding gradient problem. *CoRR*, 2012.

[62] R. Pascanu, G. Montufar, G. Montúfar, and Y. Bengio. On the number of response regions of deep feed forward networks with piece-wise linear activations, 2014.

[63] T. Peng and M. Sarazen. Iclr 2019 | 'fast as adam & good as sgd'—new optimizer has both, 2019.

[64] M. Poggi, F. Aleotti, F. Tosi, and S. Mattocia. Towards real-time unsupervised monocular depth estimation on cpu, 2018.

[65] J. D. Rodriguez, A. Perez, and J. A. Lozano. Sensitivity analysis of k-fold cross validation in prediction error estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32, 2010.

[66] S. Ruder. An overview of gradient descent optimization algorithms. *CoRR*, 2016.

[67] 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)*, 2015.

[68] S. Saito, T. Li, and H. Li. Real-time facial segmentation and performance capture from rgb input. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *Computer Vision – ECCV 2016*. Springer International Publishing, 2016.

[69] A. Sawant. Computer vision market 2019 global trends, emerging technologies, size, industry growth and segments by forecast to 2023, 2019.

[70] D. Scherer, A. Müller, and S. Behnke. Evaluation of pooling operations in convolutional architectures for object recognition, 2010.

[71] J. Schmidhuber. History of computer vision contests won by deep cnns on gpu, 2017.

[72] T. Sejnowski, M. Bartlett, and M. Mozer. Nips 2018 competition track, 2019.

[73] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, 2014.

[74] M. Sugiyama, M. Krauledat, and K.-R. Müller. Covariate shift adaptation by importance weighted cross validation. *Journal of Machine Learning Research 8*, 2007.

[75] A. Valada, R. Mohan, and W. Burgard. Self-supervised model adaptation for multimodal semantic segmentation. *CoRR*, 2018.

[76] W. Wang and U. Neumann. Depth-aware cnn for rgb-d segmentation. *CoRR*, 2018.

[77] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht. The Marginal Value of Adaptive Gradient Methods in Machine Learning. *arXiv e-prints*, 2017.

[78] Z. Wojna, V. Ferrari, S. Guadarrama, N. Silberman, L. Chen, A. Fathi, and J. R. R. Uijlings. The devil is in the decoder. *CoRR*, 2017.

[79] B. Xu, N. Wang, T. Chen, and M. Li. Empirical evaluation of rectified activations in convolutional network. *CoRR*, 2015.

[80] R. Yasrab. Ecru: An encoder-decoder based convolution neural network (cnn) for road-scene understanding. *Journal of Imaging*, 2018.

[81] F. Yu, W. Xian, Y. Chen, M. Liao, V. Madhavan, and T. Darrell. Bdd100k: A diverse driving video database with scalable annotation tooling, 2018.

[82] S. Zhang and D. Metaxas. Large-scale medical image analytics: Recent methodologies, applications and future directions. *Medical Image Analysis*, 33, 2016.

# A

# Confusion matrices

The confusion matrices for the different networks evaluated on Cityscapes.

**Table A.1:** Confusion matrix for DF-SegNet trained on Cityscapes. The diagonal shows the proportional correct classified pixels and the rows the proportional of the misclassified pixels for each class. *Note that this is not IoU accuracy, the relationship is explained in subsection 2.7.2*

| | | **Predicted class** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Actual class** | | Void | Road | Sidewalk | Building | Pole | Vegetation | Terrain | Sky | Person | Car |
| | Void | **63.3** | 5.3 | 4.0 | 16.5 | 0.0 | 4.4 | 0.5 | 0.5 | 0.7 | 4.8 |
| | Road | 1.7 | **95.6** | 1.8 | 0.0 | 0.0 | 0.0 | 0.1 | 0.0 | 0.0 | 0.7 |
| | Sidewalk | 9.5 | 17.8 | **68.7** | 1.5 | 0.0 | 0.1 | 0.9 | 0.0 | 0.1 | 1.4 |
| | Building | 3.6 | 0.0 | 0.5 | **90.4** | 0.0 | 3.7 | 0.0 | 0.6 | 0.2 | 1.1 |
| | Pole | 20.3 | 0.6 | 6.3 | 46.5 | **0.0** | 19.6 | 0.7 | 1.6 | 1.3 | 3.0 |
| | Vegetation | 2.2 | 0.1 | 0.2 | 6.7 | 0.0 | **89.2** | 0.6 | 0.5 | 0.1 | 0.6 |
| | Terrain | 14.4 | 6.7 | 9.3 | 0.8 | 0.0 | 11.0 | **56.0** | 0.0 | 0.1 | 1.6 |
| | Sky | 0.5 | 0.0 | 0.0 | 5.8 | 0.0 | 2.0 | 0.0 | **91.7** | 0.0 | 0.0 |
| | Person | 40.8 | 3.4 | 2.9 | 19.6 | 0.0 | 3.3 | 0.1 | 0.0 | **14.5** | 15.3 |
| | Car | 4.2 | 2.4 | 0.2 | 3.0 | 0.0 | 0.9 | 0.1 | 0.0 | 0.3 | **88.9** |

**Table A.2:** Confusion matrix for D-SegNet trained on Cityscapes. The diagonal shows the proportional correct classified pixels and the rows the proportional of the misclassified pixels for each class. *Note that this is not IoU accuracy, the relationship is explained in subsection 2.7.2*

| | | **Predicted class** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Actual class** | | Void | Road | Sidewalk | Building | Pole | Vegetation | Terrain | Sky | Person | Car |
| | Void | **62.5** | 5.3 | 4.0 | 18.0 | 0.0 | 4.1 | 0.4 | 0.5 | 0.9 | 4.2 |
| | Road | 1.8 | **95.5** | 1.8 | 0.1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.7 |
| | Sidewalk | 10.1 | 20.8 | **64.6** | 2.3 | 0.0 | 0.1 | 0.8 | 0.0 | 0.3 | 1.0 |
| | Building | 4.4 | 0.1 | 0.5 | **89.9** | 0.0 | 3.1 | 0.0 | 0.7 | 0.2 | 1.2 |
| | Pole | 20.8 | 1.0 | 6.6 | 47.0 | **0.1** | 17.6 | 0.6 | 2.1 | 1.3 | 3.1 |
| | Vegetation | 3.0 | 0.0 | 0.2 | 7.2 | 0.0 | **87.6** | 0.6 | 0.6 | 0.0 | 0.7 |
| | Terrain | 18.0 | 5.3 | 9.1 | 1.0 | 0.0 | 12.3 | **52.7** | 0.0 | 0.1 | 1.4 |
| | Sky | 0.5 | 0.0 | 0.0 | 4.9 | 0.0 | 1.8 | 0.0 | **92.7** | 0.0 | 0.1 |
| | Person | 32.3 | 6.9 | 2.8 | 22.8 | 0.0 | 2.7 | 0.1 | 0.0 | **19.2** | 13.1 |
| | Car | 5.3 | 3.4 | 0.2 | 4.0 | 0.0 | 1.0 | 0.1 | 0.0 | 0.5 | **85.6** |

**Table A.3:** Confusion matrix for D-SegNet trained on Cityscapes. The diagonal shows the proportional correct classified pixels and the rows the proportional of the misclassified pixels for each class. *Note that this is not IoU accuracy, the relationship is explained in subsection 2.7.2*

| | | **Predicted class** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Actual class** | | Void | Road | Sidewalk | Building | Pole | Vegetation | Terrain | Sky | Person | Car |
| | Void | **62.8** | 5.7 | 3.6 | 16.9 | 0.0 | 4.0 | 0.4 | 0.6 | 1.3 | 4.7 |
| | Road | 1.9 | **95.5** | 1.4 | 0.1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.9 |
| | Sidewalk | 10.9 | 19.4 | **63.5** | 2.8 | 0.0 | 0.2 | 0.9 | 0.0 | 0.9 | 1.4 |
| | Building | 3.7 | 0.1 | 0.6 | **90.7** | 0.0 | 2.9 | 0.0 | 0.6 | 0.3 | 1.1 |
| | Pole | 19.0 | 0.9 | 6.6 | 47.9 | **0.2** | 17.3 | 0.5 | 1.9 | 2.8 | 2.9 |
| | Vegetation | 2.5 | 0.0 | 0.2 | 7.2 | 0.0 | **88.2** | 0.5 | 0.5 | 0.1 | 0.7 |
| | Terrain | 17.0 | 5.4 | 8.0 | 1.1 | 0.0 | 15.3 | **50.4** | 0.0 | 0.3 | 2.5 |
| | Sky | 1.1 | 0.0 | 0.0 | 5.4 | 0.0 | 1.9 | 0.0 | **91.5** | 0.0 | 0.0 |
| | Person | 29.6 | 5.6 | 3.9 | 18.5 | 0.0 | 2.8 | 0.1 | 0.0 | **26.1** | 13.5 |
| | Car | 4.5 | 3.3 | 0.2 | 2.9 | 0.0 | 0.8 | 0.1 | 0.0 | 0.7 | **87.6** |

**Table A.4:** Confusion matrix for EF-SegNet trained on Cityscapes. The diagonal shows the proportional correct classified pixels and the rows the proportional of the misclassified pixels for each class. *Note that this is not IoU accuracy, the relationship is explained in subsection 2.7.2*

| | | **Predicted class** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Actual class** | | Void | Road | Sidewalk | Building | Pole | Vegetation | Terrain | Sky | Person | Car |
| | Void | **57.8** | 8.1 | 3.9 | 18.6 | 0.0 | 4.7 | 0.3 | 0.5 | 0.0 | 6.1 |
| | Road | 2.6 | **94.2** | 2.2 | 0.1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 |
| | Sidewalk | 9.2 | 31.6 | **54.3** | 1.4 | 0.0 | 0.1 | 0.5 | 0.0 | 0.0 | 2.8 |
| | Building | 3.8 | 0.1 | 0.4 | **88.8** | 0.0 | 4.6 | 0.0 | 0.8 | 0.0 | 1.6 |
| | Pole | 18.3 | 1.8 | 6.1 | 48.3 | **0.0** | 17.9 | 0.6 | 1.8 | 0.0 | 5.2 |
| | Vegetation | 2.8 | 0.1 | 0.4 | 10.2 | 0.0 | **84.5** | 0.5 | 0.6 | 0.0 | 0.8 |
| | Terrain | 17.2 | 9.8 | 25.4 | 1.4 | 0.0 | 12.2 | **31.5** | 0.0 | 0.0 | 2.5 |
| | Sky | 0.9 | 0.0 | 0.0 | 7.0 | 0.0 | 2.4 | 0.0 | **89.7** | 0.0 | 0.0 |
| | Person | 34.3 | 8.1 | 3.4 | 31.0 | 0.0 | 3.9 | 0.1 | 0.0 | **0.0** | 19.3 |
| | Car | 6.8 | 4.4 | 1.1 | 6.0 | 0.0 | 1.2 | 0.1 | 0.0 | 0.0 | **80.5** |

# B

# Predictions

Predictions on the Cityscapes and KITTI dataset.

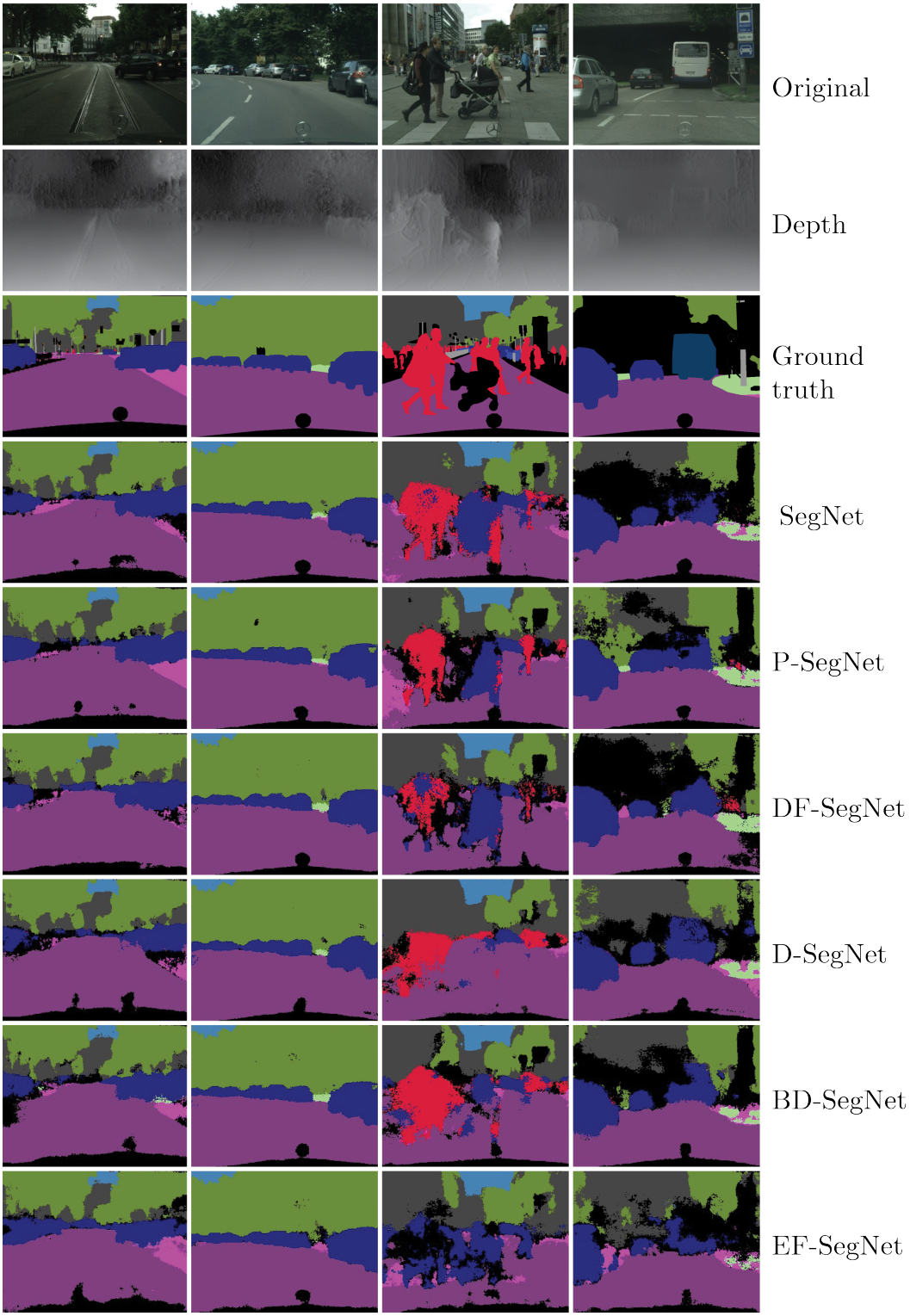



**Figure B.1:** Predictions of four images from Cityscapes. From top to bottom: Original image, corresponding depth map, pixel-wise classified ground truth, pixel-wise predictions from the six different models.

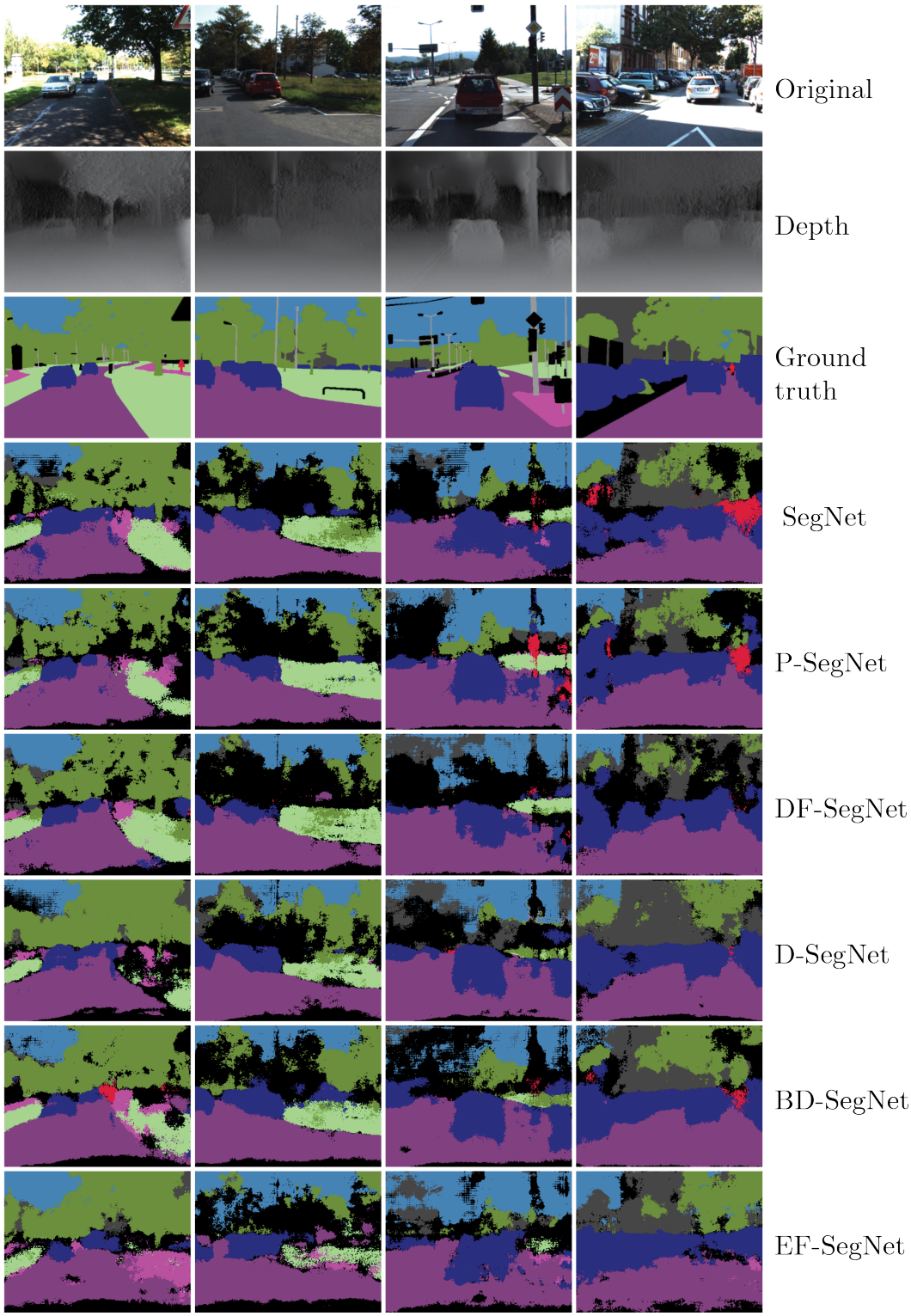
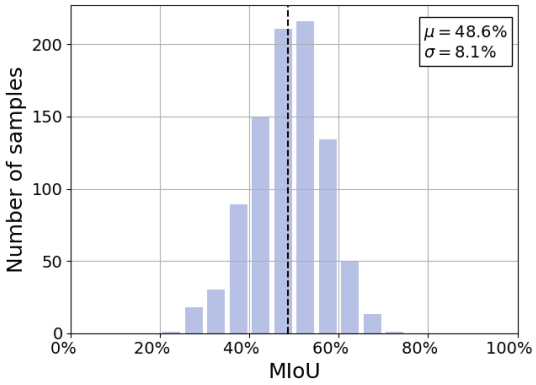



**Figure B.2:** Predictions of four images from KITTI. From top to bottom: Original image, corresponding depth map, pixel-wise classified ground truth, pixel-wise predictions from the six different models.

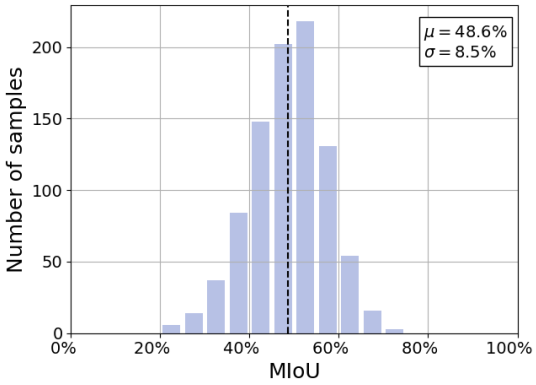
# C

# Distributions

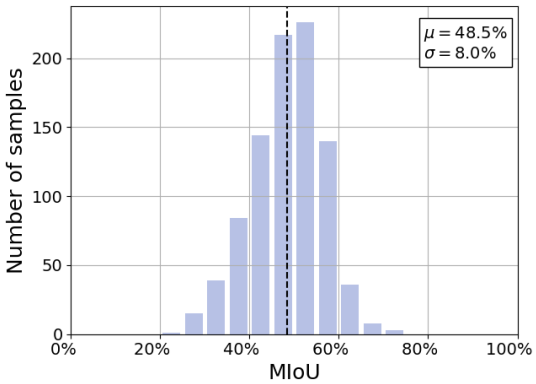
The distribution of the MIoU score for all the models on Cityscapes and the remaining classes not mentioned in the result for SegNet and P-SegNet.



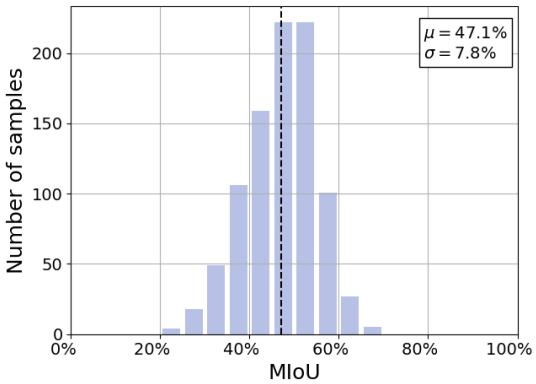

**(a)** SegNet



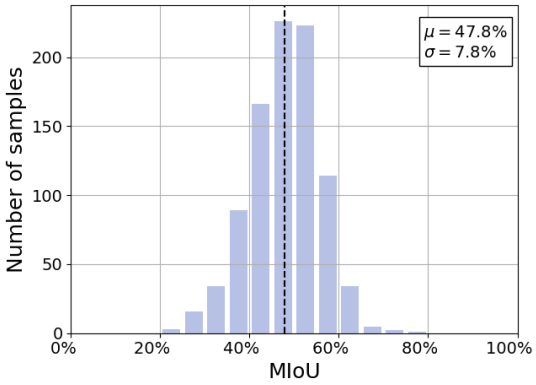

**(b)** P-SegNet



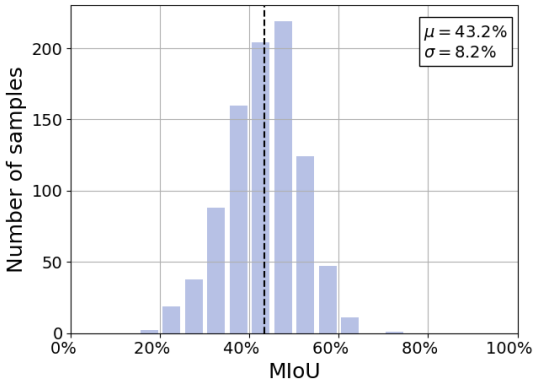

**(c)** DF-SegNet




**(d)** D-SegNet

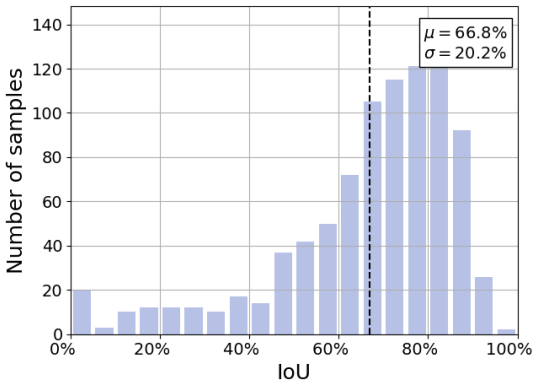



**(e)** BD-SegNet

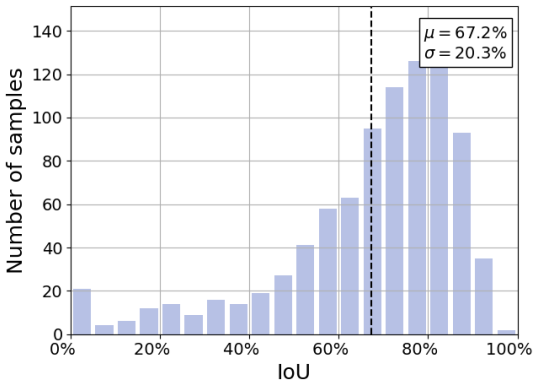



**(f)** EF-SegNet

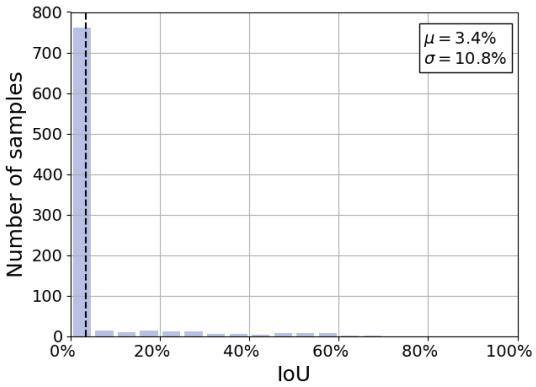
**Figure C.1:** Distribution of MIoU on the Cityscapes dataset for all the models



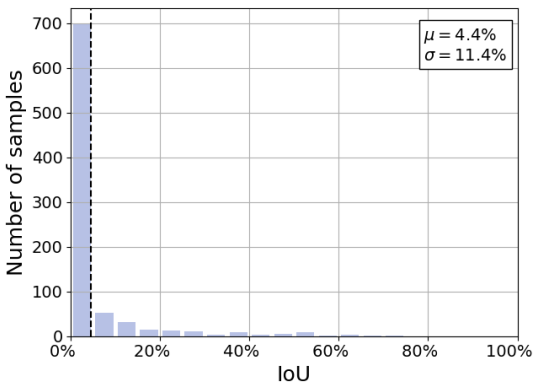

**(a)** Class *Building* on SegNet



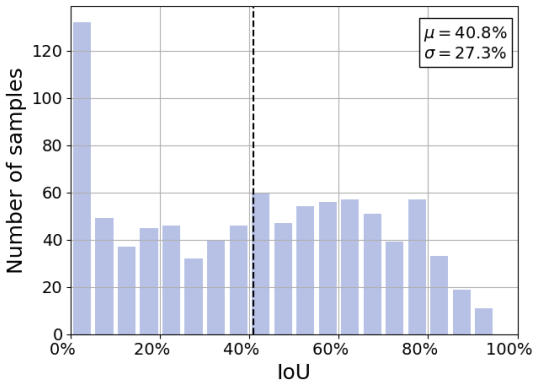

**(b)** Class *Building* on P-SegNet



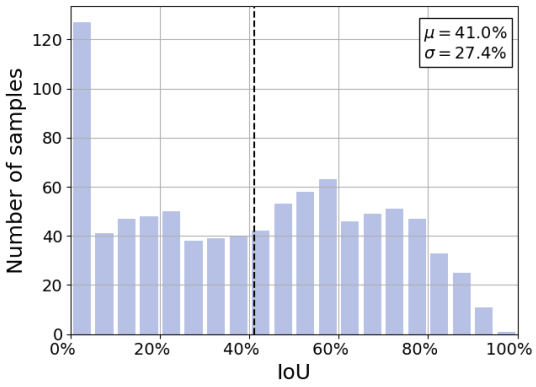

**(c)** Class *Person* on SegNet




**(d)** Class *Person* on P-SegNet

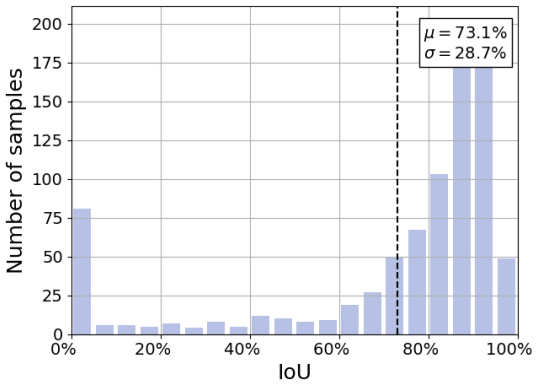



**(e)** Class *Sidewalk* on SegNet

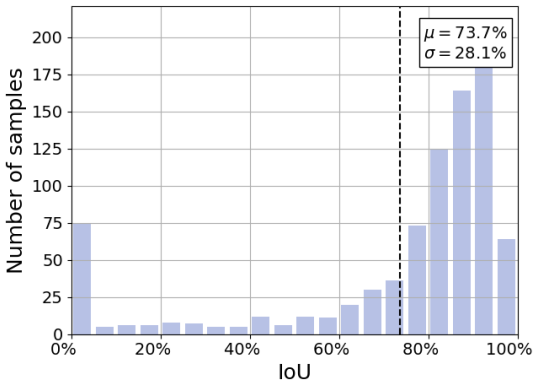



**(f)** Class *Sidewalk* on P-SegNet

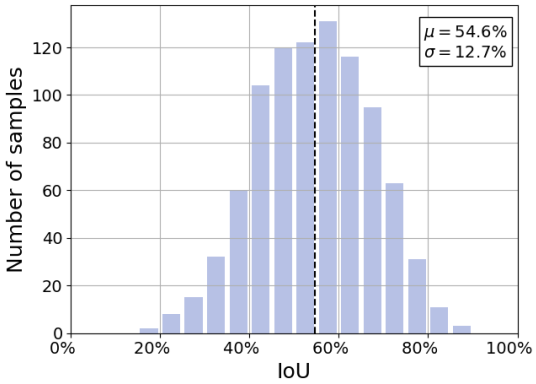
**Figure C.2:** Distribution of the IoU on the classes *Building*, *Person* and *Pole* for SegNet and P-SegNet



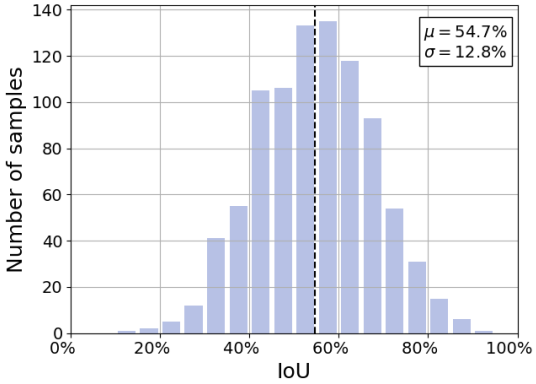

**(a)** Class *Sky* on SegNet



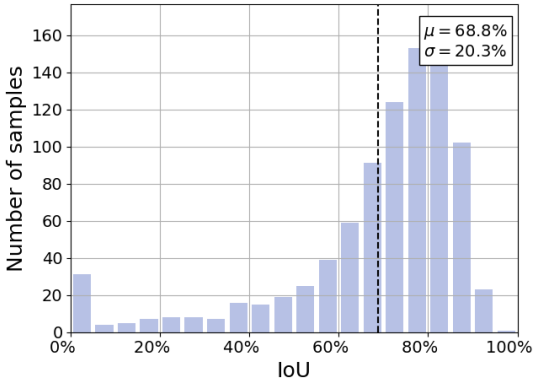

**(b)** Class *Sky* on P-SegNet



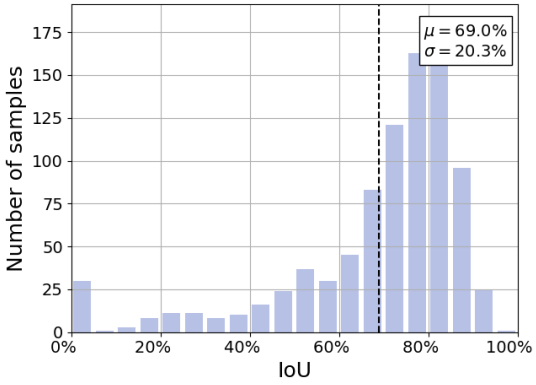

**(c)** Class *Void* on SegNet




**(d)** Class *Void* on P-SegNet




**(e)** Class *Vegetation* on SegNet




**(f)** Class *Vegetation* on P-SegNet

**Figure C.3:** Distribution of the IoU on the classes *Sky*, *Void* and *Vegetation* for SegNet and P-SegNet