Monocular depth estimation using deep convolutional neural networks

Susanna Larsson
Master of Science Thesis in Computer Vision

Monocular depth estimation using deep convolutional neural networks:
Susanna Larsson
LiTH-ISY-EX--19/5234--SE

Supervisor:  Mikael Persson
            ISY, Linköpings universitet

Viktor Ringdahl
            SAAB DYNAMICS AB

Examiner:    Michael Felsberg
            ISY, Linköpings universitet

Computer Vision Laboratory
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Copyright © 2019 Susanna Larsson
Abstract

For a long time stereo-cameras have been deployed in visual Simultaneous Localization And Mapping (SLAM) systems to gain 3D information. Even though stereo-cameras show good performance, the main disadvantage is the complex and expensive hardware setup it requires, which limits the use of the system. A simpler and cheaper alternative are monocular cameras, however monocular images lack the important depth information. Recent works have shown that having access to depth maps in monocular SLAM system is beneficial since they can be used to improve the 3D reconstruction. This work proposes a deep neural network that predicts dense high-resolution depth maps from monocular RGB images by casting the problem as a supervised regression task. The network architecture follows an encoder-decoder structure in which multi-scale information is captured and skip-connections are used to recover details. The network is trained and evaluated on the KITTI dataset achieving results comparable to state-of-the-art methods. With further development, this network shows good potential to be incorporated in a monocular SLAM system to improve the 3D reconstruction.
Acknowledgments

I would like to thank Saab Dynamics in Linköping for giving me the opportunity to work on this master thesis and for providing the resources it required. Further on, I like to thank my two supervisors, Viktor Ringdahl at Saab and Mikael Persson at LiU, for supporting me, giving me advice and answering all my questions during the thesis work. Finally, I want to thank my examiner, Michael Felsberg, for his valuable and quick feedback on the report and the master thesis work.

Linköping, 2019
Susanna Larsson
Contents

3.3.2 Data preprocessing and augmentation ................. 31
3.3.3 Mask function ........................................ 31
3.3.4 Weight initialization and frozen layers ............... 32
3.3.5 Optimization ......................................... 32
3.4 Evaluation .................................................. 33

4 Results ......................................................... 35
4.1 Modified network architectures .......................... 35
4.2 Training settings .......................................... 36
4.3 Quantitative results ....................................... 37
  4.3.1 Network architectures ................................. 37
  4.3.2 Changed training settings .............................. 39
4.4 Qualitative results ........................................ 40
4.5 Generalization to CityScapes dataset .................... 47

5 Discussion ...................................................... 49
5.1 Discussion .................................................. 49
  5.1.1 Method .................................................. 49
  5.1.2 Results .................................................. 52
5.2 Future work ............................................... 55

6 Conclusion ..................................................... 57

Bibliography .................................................... 59
List of Figures

1.1 The input image, $x$, is sent through the complex function $F$, returning the corresponding depth map, $y$. ........................................ 2

2.1 A simple example of an artificial neural network consisting of one input layer (blue nodes), two hidden layers (orange nodes) and an output layer containing one neuron (green node). The connections between the nodes represent the weights. ......................... 6

2.2 The process of how the output of a neuron, $y_k$, is generated. ..... 6

2.3 Convolving a $3 \times 3$ kernel over a $4 \times 4$ input using stride 1 and no zero-padding. The output is smaller than the input, but if zero-padding is used, an output with the same size as the input can be obtained. ........................................... 8

2.4 Atrous convolution. Convolving a $3 \times 3$ kernel over a $7 \times 7$ input with a dilation factor of 2. ........................................ 9

2.5 ReLU and PReLU activation functions. Here, $x$ is the input and the coefficient $a$ is a learnable parameter that is updated during the training process. ........................................... 10

2.6 Max- and average pooling with a filter of size $2 \times 2$ over an $4 \times 4$ feature map using stride 2 and no zero-padding. The result is an output of size $2 \times 2$. ........................................... 11

2.7 The $L_1$ and $L_2$ losses as function of the prediction error, $y - \hat{y}$. . . 12

2.8 The concept of underfitting, appropriate fitting and overfitting, (a) A linear function is fit to the data resulting in underfitting. (b) An appropriate lower order polynomial is fit to the data and generalizes well to new data. (c) A high order polynomial function is fit to the data with overfitting as result. ................................. 15

2.9 Example of a synced and rectified RGB image from the KITTI dataset [18]. ........................................... 22

2.10 Corresponding depth map, obtained from [42], to the image shown in 2.9. The image has been processed to present the depth in meters and the color of each pixel represents the depth value. Yellow represents large values and purple small values. Invalid pixels that lack depth information have a value of zero and are represented as black. Note that the depth map do not contain any valid values for the upper part. ........................................... 23
3.1 Baseline architecture. .......................... 26
3.2 Modified network architecture incorporating low level feature maps from the encoder to the decoder. Feature maps obtained from the first ResNet block are concatenated with the feature maps obtained from the third upsampling block. ....................... 27
3.3 Modified network architecture incorporating the ASPP-module. The ASPP-module is added after the last ResNet block. Note that only three upsampling blocks are used when adding the ASPP-module. 28
3.4 The ASPP-module. Convolutions with different dilation rates and global average pooling are performed in parallel. The resulting feature maps are concatenated and sent through another convolution, batch normalization and ReLU activation. ........................... 29

4.1 The validation loss of the different network architectures presented in table 4.1 as a function of the number of epochs. ....................... 38
4.2 The output predictions of the different network architectures on test image number 1. Note how network (c) and (h) more of less fail to detect the traffic sign to the right. Network (f), (g) and (i) more accurately predict the sky. ....................... 41
4.3 The output predictions of the different network architectures on test image number 2. All networks struggle to find and separate the cars in the image. .......................... 42
4.4 Qualitative results of the network architectures on test image number 3. Note how network (d)-(g) and (i) manage to find the round shape of the traffic sign to the left better than network (c) and (h). 43
4.5 Qualitative results of the network architectures on test image number 4. All networks struggle with finding the correct shape of the truck and the trees to the right. Network (f)-(i) manage to more accurately predict the top part of the image. .......................... 44
4.6 Qualitative results of the network architectures on test image number 5. All networks fail to detect the person in front of the car aswell as the person in the bottom left corner. ....................... 45
4.7 The output predictions of Mod-ASPP-LLp with different training settings. Note the sharper boundaries and the more accurate shapes in (d) and (e) compared to (c). ....................... 46
4.8 Qualitative results of the best performing network, DepNet, on the CityScapes dataset [11]. Note how the network has difficulties with reflections in car windows and close objects. ....................... 47
List of Tables

3.1 The total number of images in each set. ......................... 30

4.1 Overview of the modified network architectures and which modifications that were made to each of them. The table states the network name, the feature extractor, whether an ASPP-module was incorporated or not, the activation function, the number of sets of low level (LL) feature maps incorporated from the encoder to the decoder and the number of upsampling blocks. Also, the number of total parameters to update for each network is shown. ........ 35

4.2 Overview of the training of the different network architectures. The table states the learning rate, batch size, at which epoch the learning rate was lowered a factor 10 and the input (and consequently also the output) size. ......................... 36

4.3 Overview of the changed training settings conducted on network Mod-ASPP-LLp. .......................... 37

4.4 Quantitative results of the network architecture on the KITTI dataset, using the split proposed by Eigen et al. [15]. The table shows the average error metrics over the entire test set. The error metrics were calculated in meters at the input image resolution and only the pixels with valid corresponding ground truths were taken into account. The best values are marked in bold and the next best in cursive. .......................... 38

4.5 The average error metrics over the entire test dataset for network Mod-ASPP-LLp with changed training settings according to table 4.3. The error metrics were calculated in meters at the input image resolution, which differed between the training settings. Only the pixels with valid corresponding ground truths were taken into account. The best values are marked in bold. .......................... 39

4.6 Results on the error metrics of some depth estimation methods and the proposed network, DepNet. Note that the training and evaluation procedure varies slightly between the methods. The best results are marked in bold. .......................... 40
Imagine an autonomous car that is supposed to transport its passengers safely through a city. For the car to be able to navigate through the city without crashing into obstacles, it somehow has to observe the environment and detect existing objects. At the same time, the car has to know its own position relative to the observed environment or how else would it be able to avoid the obstacles it has detected? Simultaneous Localization and Mapping, also known as SLAM, is a technique that allows for this.

SLAM aims to create a map, either in 2D or 3D, of an unknown environment and at the same time using the same map for navigation [37]. This work focuses on visual SLAM in which cameras are used to observe the environment, but in practice a variety of different sensors can be used. By deploying stereo-cameras in the SLAM system, a true metric scaled 3D map is obtained since the depth can be calculated through triangulation. However, the main disadvantage with the stereo-cameras is the complex and expensive hardware setup and the careful calibration [43] it requires. By using monocular cameras instead, the setup can be made simpler and cheaper. When only a single monocular camera is used, the SLAM system is often referred to as mono-SLAM [12].

In mono-SLAM there are two main issues. The first is that the 3D map initially is created to an unknown scale factor, which means that the relationship between points in the map is correct but the relationship with the real world is unknown. Thus, there is no way, from within the SLAM system, to know if it is exploring a real city environment or a perfectly scaled model of one. The second issue is that the scale might drift over time and create a geometrically inconsistent map [40], a phenomena known as scale drift.
One way to solve these issues is by small-baseline stereo matching between adjacent frames. Assuming the camera translates over time, adjacent frames can be treated as stereo images and the baseline can be found during an initial phase. A crucial problem is however that pure camera rotations give no baseline and that the metric scale only is obtained once, thus the system will still be prone to scale drift. Another way to solve the pre-mentioned issues is by introducing an external scale reference, i.e. an a-priori knowledge about a set of pre-defined objects with known size, an approach which nevertheless fails in the absence of these objects [41]. Finding a way to obtain depth maps of the images would be beneficial since they can be used, similar as in the stereo-camera case, to update the inconsistent map and provide the true metric scale.

The task of estimating depth maps from monocular images can be seen as a complex high dimensional function where the input is an image and the output is the corresponding depth map. This is illustrated in figure 1.1. The human brain has an incredible ability to understand visual scenes and can solve this function by taking advantage of monocular cues in the image such as vanishing points, lighting, shading, perspective and relative scaling [19]. The monocular cues, learned experiences and knowledge about the world and size of familiar objects, are processed in the brain and a 3D perception of the scene is obtained. This is an easy task for humans, even people with monocular vision (only one eye) manage to do this. However, implementing a system capable of using monocular cues to estimate depth is a challenging task [29].

Artificial neural networks are types of deep learning algorithms that aim to mimic the brain [20] and its capacity to learn complex tasks by modelling the structure of the neural system. These methods allow systems to learn from data without being explicitly programmed and has proven useful for solving problems that are too difficult for hard-coded programs to solve [20]. In recent years, neural networks have been applied to many challenging computer vision problems such as object detection and segmentation, image classification and image reconstruction. The results are promising, in certain tasks the networks even outperform us humans [21].

Neural networks work by feeding input data to the network and returning some output data. How the input is mapped to the output is determined by the
1.1 Motivation

Networks internal weights, also known as its parameters. The process of finding the suitable weights is referred to as training, or optimization, of the network. Supervised neural networks are trained on inputs with corresponding desired outputs, or ground truths, that show the network how it should behave. The goal is for the network to learn to return accurate outputs, or predictions, given an input it has not seen before [20].

This thesis work aims to use supervised neural networks to estimate depth from monocular images to improve the 3D reconstruction in mono-SLAM systems. Advances in deep learning in combination with development of faster, more powerful computers and the access to large dataset motivates using this approach. Furthermore, several attempts to estimate depth from monocular images using neural networks have been made in the recent years [5, 15–17, 19, 23, 27, 29] showing promising results.

1.1 Motivation

Having access to depth maps in SLAM-systems has shown beneficial [41, 43] since they provide the true metric scale and at the same time can be used to reconstruct geometrically consistent maps. Also in other applications, such as robot navigation, 3D-modelling [4] and augmented reality systems [1], accurate depth estimation is a fundamental task. Being able to robustly predict depth maps from monocular images would make these systems more robust, simpler and cheaper. Further, this thesis work gives an indication of the possibility of replacing stereo-cameras with cheaper, more light weighted and lower power consuming monocular cameras.

1.2 Aim

This master thesis work aims at implementing, training and evaluating a deep convolutional neural network that predicts depth from monocular RGB images. This work investigates the possibility of modifying a state-of-the-art network to improve its performance. The implemented method is evaluated quantitatively according to appropriate error metrics but also qualitatively through visual inspection. The method is trained and evaluated on the public KITTI depth dataset.

1.3 Research questions

The aims of the thesis work can be broken down into three research questions as follows:

- Which kind of neural network architectures have previously been used for estimating depth from monocular images and what makes them suited for this particular task?
• How does the proposed network perform compared to similar state-of-the-art methods?

• How do the implemented network architectures, assuming training on the same dataset, differ in terms of training time and performance?

### 1.4 Limitations

No post-processing steps to improve the output predictions will be included in the implemented methods. This is due to the desire to evaluate the architecture of the neural networks and not the post-processing.

Computational resources and time constraints limit the search for the optimal hyperparameters and augmentation techniques. Therefore only the most significant hyperparameters are varied, while other choices are made based on the literature and results from previous related works.
In this chapter, theory about neural networks that is relevant to this thesis and related work are presented.

2.1 Artificial neural networks

Artificial Neural Networks (ANNs) are inspired by the biological neural system and consist of multiple nodes, so called neurons, which are connected to each other. A general ANN has an input layer, one or multiple hidden layers and an output layer. Each layer contains a number of neurons that are connected to the neurons in the adjacent layers. If the layer is a fully connected layer, each neuron in each layer is connected to all neurons in the adjacent layers. Artificial neural networks that have two or more hidden layers are referred to as deep neural networks, DNNs [32]. The required number of hidden layers and neurons depends on the complexity of the task and the application. The general structure of an ANN is shown in figure 2.1.
Figure 2.1: A simple example of an artificial neural network consisting of one input layer (blue nodes), two hidden layers (orange nodes) and an output layer containing one neuron (green node). The connections between the nodes represent the weights.

Similar to a biological nerve cell, an artificial neuron receives inputs from other neurons and transmits an output to some other neurons. How much a specific input affects the output is described by certain weights, which can be thought of as the strength of the connections. The weights tell us to what extent a specific input exhibits or inhibits the output [13]. The output is thus a weighted sum of all the connected input neurons. The output is then transformed by sending this linear combination through an activation function (revised in section 2.2.2) to generate the final output that is passed on to the other neurons. This is illustrated in figure 2.2.

Figure 2.2: The process of how the output of a neuron, \( y_k \), is generated.
The process of how the output of a neuron is generated is expressed mathematically according to (2.1).

\[ y_k = \sigma \left( \sum_n w_{kn} * x_n + b_k \right) \]  

(2.1)

Here, \( x_n \) is the input from the \( n^{th} \) neuron in layer \( l - 1 \), \( y_k \) the output of the \( k^{th} \) neuron in layer \( l \), \( w_{kn} \) is the weight connecting neuron \( x_n \) to \( y_k \), \( b_k \) is an overall bias term for neuron \( y_k \) and \( \sigma \) is an activation function.

In supervised learning, i.e. if the input has a corresponding ground truth, the hidden layers of the network aim at learning a mapping between these. By combining multiple neurons with appropriate weights and biases, complex functions can be learned, resulting in the desired behaviour of the network.

The error between the ground truth and the predicted output is used to update the learnable parameters (the weights and biases), in such a way that the error is minimized. The weights are adjusted iteratively based on gradient decent during the training process [32], which is described further in section 2.3.2. The error to minimize during the training is calculated using a loss function, which is described in section 2.3.1.

### 2.2 Convolutional neural networks (CNNs)

Convolutional neural networks (CNNs) use convolutional operations in one or more of the layers. CNNs are particularly well adapted for solving image related problems [32].

The three main components of a CNN are convolutions, activation functions and pooling [20], and a CNN typically contains multiple layers involving these operations. The main components are described in section 2.2.1-2.2.3. CNNs often involve two other types of operations called batch normalization and drop-out, see section 2.3.3. During the training process, these operations allow the network to learn the important features needed for solving the task [2].

#### 2.2.1 Convolutional layers

In the convolutional layer, the input, which often is an image, is convolved with a filter that slides over the entire image. In this way, each neuron in the hidden layer will be connected to different regions of the neurons in the previous layer. The region size is determined by the filter size and is referred to as the receptive field [2]. How many pixels the filter moves in each new step is called the stride [32]. An illustration of a regular convolution is shown in figure 2.3.

The result of the convolution is called a feature map, since it catches features, or patterns, in the image. A feature can for example be a horizontal edge or line
but also a more complex structure. The weights and biases of the filters are learned during the training and determine which kind of features to extract. The same weights and bias are shared between all neurons in the layer, which is referred to as parameter sharing. This means that the same filter is applied across the entire image to detect the same feature but at different locations. This is motivated by the fact that features that are useful in one region probably are useful in other regions of the image as well [32].

**Figure 2.3:** Convolving a 3×3 kernel over a 4×4 input using stride 1 and no zero-padding. The output is smaller than the input, but if zero-padding is used, an output with the same size as the input can be obtained.

**Atrous convolutions**

Atrous convolution was introduced by Chen et al. [7] to enlarge the receptive field while maintaining the same number of parameters. In conventional CNNs a larger receptive field can be achieved in two ways. The first is to use pooling layers to reduce the spatial size followed by convolution. The second way is simply to use larger filters. The problem with using pooling layers is that the resolution of the feature map is decreased and using larger filters results in more parameters, thus making the network more complex to train.

To overcome these problems, Chen et al. [7] remove the downsampling pooling layers and instead perform convolution with upsampled filters, where zeros are inserted between the filter values. In this way the receptive field is enlarged while the number of parameters stay the same, since only the non-zero values of the filter are taken into account. This kind of convolution is referred to as atrous, or dilated, convolution and is illustrated in figure 2.4.

How much the filter is upsampled, and consequently how much the receptive field is enlarged, is defined by the dilation rate. A rate = 1 corresponds to a regular convolution and rate = 2 means that the filter is upsampled a factor 2 and thus one zero is inserted between each of the weights. By altering the dilation rate, the receptive field can be arbitrary enlarged and the resolution at which the feature map is obtained can be controlled [7]. Chen et al. [7] show that
a larger receptive field is particularly desirable for dense prediction tasks since a larger global context of the image can be incorporated.

![Atrous convolution](image)

**Figure 2.4**: Atrous convolution. Convolving a $3 \times 3$ kernel over a $7 \times 7$ input with a dilation factor of 2.

A challenge in dense prediction tasks is the fact that objects exist at different scales and the network must be able to handle the scale variability to make accurate predictions. Atrous spatial pyramid pooling (ASPP) was proposed in [7] to capture image context and objects at different scale by performing multiple parallel atrous convolutions with different dilation rates.

### 2.2.2 Activation function

The obtained linear combination of inputs is sent through an activation function that maps the values to another range and introduces non-linearities [33] to the otherwise linear model described in (2.1). This allows the model to learn high order polynomials between the input and ground truth, which is necessary since the underlying problem most likely is not linear.

There are different kinds of activation functions and the most commonly used in CNNs today is ReLU [32] but also its variant PReLU is a popular choice. These functions are shown in figure 2.5 and described in the sections below.
Rectified linear unit (ReLU)

The rectified linear unit, or ReLU, is a non-linear activation function widely used due to its simplicity and reliability [33]. For an input $x$, ReLU is defined as

$$f(x) = \max(0, x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases} \quad (2.2)$$

The advantage with ReLU is the fast and easy computations it leads to, which results in faster training [30]. Also the problem with vanishing gradients as $x$ increases, is avoided since the gradient for $x \geq 0$ is constant.

As seen in figure 2.5, ReLU only allows positive input values to be activated while negative values are mapped to zero and thus give no activation. During the backpropagation (discussed in section 2.3.2) the parameters of the non-activated, or dead, neurons will not be updated, which hinders the learning [33].

Parametric Rectified Linear Unit (PReLU)

To overcome the problem with dead neurons and to improve model fitting, a variant of ReLU called parametric ReLU, or PReLU, was introduced by He et al. [21]. This activation function allows an activation even for negative inputs, which can be seen in figure 2.5. PReLU is defined in (2.3) in which $a$ is a coefficient controlling the negative slope of the function.

$$f(x_i) = \begin{cases} x_i, & \text{if } x_i > 0 \\ a_i x_i, & \text{if } x_i \leq 0 \end{cases} \quad (2.3)$$

If $a = 0$ PReLU becomes the standard ReLU. The coefficient $a$ is a parameter that is learned during the training. It can either be learned channel-wise or shared between channels, which is indicated by the subindex $i$ [21].
2.2.3 Pooling layers

After the feature map is sent through the activation function, a pooling layer is typically applied to reduce its spatial size. This reduces the number of parameters the network needs to learn and thus improves the computational efficiency of the network [20]. It also increases the receptive field and makes the network invariant to small translations of the input. The pooling layers have no learnable parameters, but instead a fixed function is applied separately on each of the feature maps [2]. During this operation a small filter, typically of size 2x2 or 3x3, slides over the feature map and returns a summary statistics of the nearby values. The most popular choices of pooling are max-pooling and average-pooling [20], illustrated in figure 2.6.

![Figure 2.6: Max- and average pooling with a filter of size 2x2 over an 4x4 feature map using stride 2 and no zero-padding. The result is an output of size 2x2.](image)

Max-pooling involves taking the maximum of the values inside the filter region. Since the same feature is detected over the whole feature map, returning the maximum value in a certain region gives an indication of whether the feature is present in that region or not. In this way the network becomes invariant to small translations of the input image, since the maximum value always is extracted within a small region. However, this also means that the exact spatial location of the feature is lost. Average-pooling is similar to max-pooling but instead returns the average of all values inside the filter region.

Since only one value is returned for a region that contains multiple values, the spatial size of the feature map is reduced. How much it is reduced, i.e the downsampling factor, is determined by the filter size and the stride. To get the desired downsampling factor, zero padding can be used.
2.3 Training a neural network

In this section the general training principles of supervised neural networks are described.

2.3.1 Loss function

The loss function is used to measure the difference between the ground truth value, \( y \), and the predicted value \( \hat{y} \). The difference is often referred to as the prediction error and gives a numerical value of how well the network is able to perform its task. In cases when the predictions are made pixelwise, the prediction error is also calculated pixelwise. During the training process, the weights and biases are updated with the objective to minimize the loss function [20]. To choose an appropriate loss function is therefore essential for the network to learn its task properly. Some common loss functions for regression problems are \( L_1 \), \( L_2 \), and the BerHu loss function [6]. These functions are shown in figure 2.7 and further described in the sections below.

\[ L_1(\hat{y}) = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i| \]  

(2.4)

Here, \( y_i \) is the ground truth value in pixel \( i \) and \( \hat{y}_i \) is the predicted value in that pixel. The loss is summed over all pixels and divided by the total number of pixels, \( N \), to get the mean value. The \( L_1 \) loss is a linear function that grows linearly with the size of the prediction error, which can be seen in figure 2.7.

Figure 2.7: The \( L_1 \) and \( L_2 \) losses as function of the prediction error, \( y - \hat{y} \).
The $L^2$ loss is also called the mean square error and is defined in (2.5) [6].

$$L^2(\hat{y}) = \frac{1}{N} \sum_{i} (y_i - \hat{y}_i)^2$$ (2.5)

Here, $y_i$ is the ground truth value in pixel $i$ and $\hat{y}_i$ is the predicted value in that pixel. The squaring of the prediction error results in a loss function with a non-linear behaviour as seen in figure 2.7. In the figure it can also be noticed that the $L^2$ loss returns a lower value than the $L^1$ loss if the prediction error is less than one but a higher value if it is larger than one. This means that the $L^2$ loss returns a higher loss value for larger prediction error, which can cause problems with outliers. However, by squaring the loss, the calculations of the gradient of the loss function during the backpropagation are simplified.

**BerHu loss**

The BerHu loss, first introduced in [34] is a reversed variant of the Huber loss. It differs from the other loss functions since it acts differently depending on the value of the absolute prediction error. The BerHu loss is defined in (2.6) [29].

$$B(\hat{y}) = \begin{cases} 
|y - \hat{y}|, & |y - \hat{y}| \leq c \\
\frac{(y - \hat{y})^2 + c^2}{2c}, & |y - \hat{y}| > c
\end{cases}$$ (2.6)

Here, $c$ is a positive real number and the function returns the $L^1$ loss if the absolute prediction error is smaller than $c$ and a scaled version of the $L^2$ loss if it is larger than $c$. In this way the linear properties of the $L^1$ and the non-linear properties of $L^2$ are implemented, which is presented in [29] to be the benefit of using the BerHu loss. The value of $c$ decides where the switch between the two loss functions happens but also scales the quadratic loss function. Often $c$ is calculated as $c = \frac{1}{2} \max_i (|y_i - \hat{y}_i|)$ as explained in [29], where $i$ indicates all pixels in each image in the current batch.

### 2.3.2 Optimization

During the training phase, the objective is to minimize the loss function [20] with respect to the weights of the network. This is referred to as optimization of the network.

**Forward and backward propagation**

The training process starts with a forward propagation where the input is passed forward through the network to produce the prediction. The prediction is evaluated against the ground truth using a loss function, which is described in section 2.3.1. The gradient of the loss function is calculated with respect to the weights and the information is sent back through the network via
backpropagation. During the backpropagation the chain rule is used to decompose the gradient into local gradients for each neuron, which tell us the impact each weight has on the loss function. The local gradients are then used to update the weights via a gradient decent method in such a way that the loss is minimized [20].

**Stochastic gradient descent**

When using gradient descent (GD), the gradient is computed over the entire training set and the weights are updated after each epoch. An epoch is complete when all training samples have been propagated forward and backward though the network. However, if the training set is large the method takes a long time to compute the gradients [32]. To speed up the training process considerably, *Stochastic-Gradient-Descent* (SGD) can be used instead [20].

SGD is an approximation of GD and a popular optimization method within deep learning. SGD computes the gradient over a set of randomly chosen samples, or a batch, instead of over the entire set. By taking the average over the batch, an approximation of the true gradient is obtained and used to update the weights accordingly [32]. The weight update using SGD is defined in (2.7).

$$\begin{equation}
w_k' = w_k - \frac{\eta}{M} \sum_i \frac{\partial F(x_i)}{\partial w_k}
\end{equation}$$

Here, $w_k$ is the current weight, $w_k'$ the updated weight, $\eta$ the learning rate, $M$ the number of samples in the batch and $F(x_i)$ the loss for the $i^{th}$ sample.

One advantage with SGD is that, even if the number of training samples increase, the computation time for each update remains the same. This allows for convergence even for large training sets [20]. Also, since the gradient is calculated over a random batch each time, the SGD is less likely to get stuck in local minimums, while the GD is more prone to this.

The learning rate is a hyperparameter that is set by the user and can have large impact on the training performance. Using a large learning rate will result in faster training since large steps are taken each time. However, if the learning rate is too large the system might overshoot the minimum and diverge instead. A smaller learning rate will cause slow convergence and there is risk of getting stuck in local minimums. In general, a good approach is to train the network with a larger learning rate for a number of epochs and then lower the learning rate [32]. Normally, several iterations, or weight updates, are necessary to make the network converge.

A popular SGD variant is the Adaptive Moment Estimation (ADAM) optimizer
2.3 Training a neural network

[28] that adapts the learning rate individually for each parameter. The method has shown great performance in terms of computational efficiency and memory requirements and is particularly suited for problems involving large datasets and high-dimensional parameter spaces.

**Overfitting and underfitting**

A desirable behaviour of a network is the ability to perform well, not only on the training data, but on data it has not seen before. The network ability to do this is called *generalization* and is usually measured by evaluating the performance on a test set containing previously unseen data [20].

A network containing too few parameters might not be able to adapt well to neither the training set nor the test set, something that is referred to as *underfitting*. With a large number of parameters, the network will be able to learn a more complex mapping between the input and output. However, this does not necessarily make it a good model since it could become too adapted to the training data and fail to generalize to the test set [32]. This phenomena is known as *overfitting* and is a central challenge in deep learning [20].

Appropriate model fitting is when the model performs well on the training set and the test set. The concept of underfitting, appropriate fitting and overfitting is illustrated in figure 2.8.

![Figure 2.8](image)

**Figure 2.8:** The concept of underfitting, appropriate fitting and overfitting. (a) A linear function is fit to the data resulting in underfitting. (b) An appropriate lower order polynomial is fit to the data and generalizes well to new data. (c) A high order polynomial function is fit to the data with overfitting as result.

Deep neural networks are more prone to overfitting due to their complexity and large amount of parameters [39]. Fortunately, there are some techniques, known as *regularization techniques*, that can be used to reduce the risk of overfitting.
Some of these techniques, used in this master thesis work, are described below in section 2.3.3.

### 2.3.3 Regularization techniques

Regularization techniques are used to improve the networks ability to generalize to unseen data. Three common regularization techniques are drop-out, batch normalization and data augmentation.

**Drop-out**

Drop-out is a regularization technique where a random subset of hidden neurons are temporarily deleted during the training phase. The deleted neurons will not have their parameters updated during the backpropagation [32]. If a neuron should be deleted or not is controlled by a hyperparameter, \( p \), that is set to a fixed value. Typically \( p = 0.5 \), which means that a hidden neuron is deleted with a probability of 50% [20].

Which neurons to delete is randomly chosen for each batch, making the network architecture slightly different during each backpropagation [32]. In this way, the neurons learn to work with a set of randomly connected neurons and are prevented from becoming too dependent on each other. This makes the network learn more robust and useful features, which reduces the risk of overfitting [39]. Drop-out is particularly useful when training deep networks where the risk of overfitting is high [32].

**Batch normalization**

During the training, the distribution of each layers inputs changes since the inputs are affected by the update of the parameters in all preceding layers. This phenomena is referred to as **covariance shift** and complicates and slows down the training by requiring low learning rate and careful weight initialization [25]. Even small changes to the parameters can be amplified through the network as it grows deeper, leading to exploding or vanishing activations. Batch normalization was introduced by Ioffe et al. [25] to reduce the covariance shift by normalizing the input to each layer on a per batch basis, hence the name. By performing batch normalization, saturated activations are prevented, the learning becomes more stable and higher learning rate can be used. It also removes the need of careful weight initialization.

**Data augmentation**

The best way to improve the ability of a network to generalize is to train it on more data [20], but obtaining training data can be a difficult, costly and time-consuming process. **Data augmentation** extends the existing dataset by altering the images in various ways. The altered images will be perceived by the network as new images and thus the dataset size is increased, which improves
the robustness of the network and reduces the risk of overfitting [20].

Operations like rotation, flipping and cropping are simple and successful augmentation strategies [36]. Also translation and scaling have proved to be efficient. Sometimes, a small amount of Gaussian noise with zero mean can be added to the input images to prevent the network from learning unuseful high frequency structures, which makes the network more robust [20].

2.4 CNN for pixelwise depth prediction

Depth estimation is a dense prediction task in which each pixel in the image is assigned a depth value. It is closely related to the task of semantic segmentation that aims at assigning each pixel a class label from a number of categories [26]. For dense prediction tasks, it is desirable to obtain an output prediction with the same spatial size as the input, so each input pixel has a corresponding prediction. This can be achieved by using an encoder-decoder network architecture. In general, the encoder extracts useful features from the input image and downsamples the feature maps. The decoder then upsamples the feature maps, recovers the image resolution and produces the pixel-wise predictions [8].

Several networks, for example SegNet [3], ENet [35] and DeepLabv3+ [9], have been designed to perform semantic segmentation following an encoder-decoder structure. Other types of networks, such as ResNet [22] and VGG [38], originally designed for image classification, do not contain a decoder. Although, the first layers of the classification networks work as encoders and can therefore be exploited as part of an encoder-decoder network. In fact, many semantic segmentation networks use pretrained image classification networks and fine-tune these to perform semantic segmentation [26]. This is true even for pixel-wise depth prediction, for example Laina et al. [29], Fu et al. [16], Jiao et al. [27] and Cao et al. [5], all use ResNet as encoder.

2.5 Related work

This section presents a number of works in which neural networks has been used to predict depth from monocular images. The KITTI dataset [18] and the most commonly used error metrics for depth prediction are also described here.

2.5.1 Depth predicting neural networks

Several attempts to solve the depth estimation problem using deep convolutional neural networks have been made in the recent years and the results are promising [5, 15–17, 19, 23, 27, 29]. Below, a number of these works are described to give an insight about the state-of-art methods and their benefits. This will later be used to motivate and to choose the implemented method.
**Multi-scale deep neural network, 2014**

One of the first successful attempts to predict depth from monocular images using deep neural networks was performed by Eigen et al. [15] with the use of a multi-scale deep neural network. The method is based on two networks, one coarse and one fine. The coarse network predicts the depth of the scene at a global level taking advantage of the entire image. The output from the coarse network, together with the original image, is passed as input to the fine network that refines the prediction by using local information. The network was pretrained on the ImageNet classification task, which was found to be slightly better than random initialization.

Eigen et al. [15] found that the fine network did not improve the error metrics significantly, but a noticeable difference could be seen in the qualitative results. The network achieved state-of-the-art results on all the error metrics for both the NYU-Depth V2 dataset and KITTI dataset.

This work shows the benefit of using information from multiple scales and that integrating the global context of the image is important to be able to take advantages of monocular cues.

**Fully Convolutional Residual Network (FCRN), 2016**

Laina et al. [29] propose a method using fully convolutional residual networks to predict depth from monocular images. Their network is based on ResNet-50 initialized with pretrained weights. They replace the fully connected layers in ResNet with novel residual upsampling modules. By removing the fully connected layers the number of learnable parameters is reduced, making their network simpler, faster and more efficient compared to the fully connected architecture.

Laina et al. [29] introduce up-convolution that involves an 2x2 unpooling, followed by a 5x5 convolution and a ReLU activation. Further, they improve their up-convolutions by introducing an additional convolution and adding a projection connection from the lower level feature map to its output. They call these up-projection blocks and this extends the idea of skip connections, used in ResNet [22], to up-convolutions.

By creating a chain of up-projection blocks, high-level information is passed through the network efficiently and at the same time the spatial size of the feature map increases. Four up-projection blocks are stacked together, yielding an up-scaling of the smallest feature map with a factor 16 in each dimension. This was found to be the best trade-off between resolution and memory usage. The BerHu loss function was used and found to be better suited than the typically used $L_2$ loss when exploiting depth maps as ground truth.

Laina et al. [29] compare different backbone network architectures: AlexNet,
VGG-16 and ResNet-50. AlexNet is outperformed by the other two networks due to its small receptive field at the last convolutional layer, which limits the input resolution and does not capture enough global information. Using VGG-16, which has a larger receptive field, results in higher accuracy on the depth prediction, although ResNet-50 gives by far the best performance on all error metrics.

The results of this work shows the benefit of using up-projections in the decoder to obtain dense output predictions of higher resolution. It also shows the potential of using ResNet as the feature extractor instead of VGG or AlexNet.

**Embedding focal length, 2018**

He et al. [23] propose a novel deep convolutional neural network embedding information about the camera’s focal length for predicting depth of monocular images. They start by showing that there exists an ambiguity between depth of the scene and the camera’s focal length and that the focal length has great influence on the accuracy of the depth estimation. The method aims to give each pixel a depth value and they use an encoder-decoder structure that is built on VGG initialized with pretrained weights followed by fully connected layers and upsampling. The focal length information is embedded in the last fully connected layers of the network. They also incorporate skip-connections, inspired by ResNet [22], to exploit features from different layers. Similar to [29] they use the BerHu loss function to exploit the advantages of both the $L_1$ and $L_2$ losses.

This work shows that providing the network with additional inputs can improve the depth predictions.

**Semantic booster and attention driven loss, 2018**

For most scenes the pixel depth values are unevenly distributed and typically have a long tail distribution. Most existing networks use loss functions that treats all pixels equally, causing models to predict the most common, often small, depth values. Jiao et al. [27] propose a method that takes the imbalanced depth distribution into account by using an attention driven loss for the network supervision. At the same time, they exploit the fact that depth and semantics in the scene share context information and can benefit from one another.

Jiao et al. [27] implement a multi-task deep convolutional neural network that consists of a depth predicting and a semantic labeling sub-network. The input is passed through a backbone encoder based on ResNet-50 pretrained on ImageNet, which outputs high-dimensional feature maps. The feature maps are sent to the sub-networks to predict the depth and semantic labels. A Lateral Sharing Unit (LSU) is used for knowledge sharing between the two tasks and the network learns this sharing strategy.

This work shows that using a loss function adapted to the dataset distribution
and incorporating semantic information can improve the depth predictions.

**Deep Ordinal Regression Network (DORN), 2018**

Fu et al. [16] introduce a Deep Ordinal Regression Network adopting a multi-scale strategy to predict depth from monocular images. They discretizes the depth values into a number of intervals using a space-increasing discretization method (SID). This method considers the fact that the uncertainty in depth prediction increases as the depth values increase, thus allowing larger errors for larger depth values. The depth prediction is cast as an ordinal regression problem, which has the properties of both regression and classification and aims at assigning each pixel into a set of ordered categories [10].

Fu et al. [16] highlight the problem with standard CNN:s in which repeated convolution and pooling operations gradually decrease the spatial size of the feature map, which is undesirable for dense prediction tasks. To overcome this, multi-layer de-convolution and skip-connections can be used, but with the cost of a more complex structure and additional memory and computational cost. Instead, Fu et al. [16] capture multi-scale information by using an atrous-spatial-pyramid-pooling (ASPP) module, first introduced in [7]. The ASPP module contains atrous convolutions with various dilation rates to enlarge and vary the receptive field of the filters. They also compare VGG-16 and ResNet-101 and found that ResNet-101 performs better than VGG-16 on all error metrics.

Fu et al. [16] lift forward the advantage of using an ASPP-module to obtain feature maps of higher resolution and that using ResNet over VGG results in better performance.

**Unsupervised CNN for single view depth estimation, 2016**

Garg et al. [17] propose a network trained in an unsupervised manner and justify their work by the weakness of supervised NN:s requiring a vast amount of annotated data. They post the depth estimation problem as an image reconstruction task using an auto-encoder setup to train on a pair of images from a stereo camera. The network predicts the inverse depth of the left image, then uses inverse warping, the disparity, and the right image to reconstruct the left image. The reconstructed image is matched to the input and during the training phase the reconstruction loss is minimized.

Their network is based on AlexNet, and similar to the work in [29], the last fully connected layers are replaced with fully convolutional layers. To get refined predictions, they employ skip-connections. To evaluate their method, the predicted disparity (inverse depth) map is up-sampled to the original image size and by using the known stereo baseline, the disparity is converted to depth.

This work lifts forward the advantage of using skip-connections to combine global and local information.
Unsupervised monocular depth estimation with left-right consistency, 2016

Another unsupervised method was introduced by Godard et al. [19], which is similar to [17] but uses a reconstruction loss with a left-right consistency. They show that only taking the reconstruction loss into account can give good image reconstruction, but results in poor depth predictions. By exploiting epipolar geometry constrains, their network can simultaneously generate disparity maps for both images (left to right and right to left) using only the left image. By enforcing consistency between the disparity maps during the training phase, the depth predictions become more robust and accurate.

This work shows how a network can be trained for depth estimation in an unsupervised manner using only pairs of stereo images.

Depth estimation as a classification task, 2017

Most depth prediction networks treat the problem as a regression task [15, 17, 19, 23, 27, 29] but there have been some attempts to cast the problem as a classification task instead, which partially was done in [16]. Cao et al. [5] propose the depth estimation problem as a pixel-wise classification task by using a fully convolutional residual network and fully connected conditional random fields (CRF:s) for post-processing. They mean that depth values are hardly exactly regressed to the ground truth and that it is easier to predict a pixels depth interval instead of the exact value. They also mention that a problem with regression is that it tends to predict the mean depth value, which can results in large error for areas that are very close or very far away. Another advantage of casting the problem as a classification task is that the confidence of a prediction can be obtained.


This work shows the potential of casting the depth estimation problem as a classification task and that ResNet yields better performance than VGG.

2.5.2 KITTI dataset

For training and evaluating depth predicting networks, the KITTI dataset [18] is a popular choice. It has been used to train most of the networks [5, 15–17, 19] mentioned in section 2.5.

The dataset consists of approximately 93 thousand images from outdoor scenes, divided into five categories; city, residential, road, campus and person. The data was recorded using a high resolution color stereo camera rig and a Velodyne 3D laser scanner (LiDAR), which captured the depth of the scene. In total, 151
sequences were recorded and for each frame and the raw data from the left and right camera are provided. A rectified version of the RGB images together with the raw LiDAR scans, are available for download\(^1\).

The resolution of the rectified RGB images varies slightly depending on the calibration parameters but are approximately 1242x375.

The raw LiDAR scans are sparse, which means they do not contain values for all pixels. However, denser depth maps were provided by Uhrig et al. [42] that derived a large scale dataset of depth annotated RGB images from the sparse scan data. They motivated their work by showing that traditional convolutional neural networks usually perform poorly when they are trained on sparse data. The provided depth maps are four times denser than the raw LiDAR scans, i.e they contain values for four times as many pixels [42].

Figure 2.9 and 2.10 show an example of an input image and the corresponding ground truth depth map from the KITTI depth dataset [42]. For all depth maps, valid values are only provided for the lower part of the image, as can be seen in figure 2.10, and invalid pixels (pixels without a depth value) has a value of zero.

\[\text{Figure 2.9: Example of a synced and rectified RGB image from the KITTI dataset [18].}\]

\(^1\)http://www.cvlibs.net/datasets/kitti/raw_data.php
2.5 Related work

![Corresponding depth map](image)

**Figure 2.10:** Corresponding depth map, obtained from [42], to the image shown in 2.9. The image has been processed to present the depth in meters and the color of each pixel represents the depth value. Yellow represents large values and purple small values. Invalid pixels that lack depth information have a value of zero and are represented as black. Note that the depth map do not contain any valid values for the upper part.

**Eigen split**

Eigen *et al.* [15] introduced a split of the KITTI dataset that has been used as reference among many depth predicting works. Eigen *et al.* split the dataset into a training, a validation and a test set and the specific split files were later made public by Godard *et al.* [19] at GitHub\(^2\). The training set consists of approximately 22 600 frames from a total of 28 different scenes and the validation set contains of 888 frames. The test set contains 697 frames from 28 different scenes, which are different from the scenes used for training och validation.

**Data augmentation**

Applying data augmentations relevant to the task is essential for the network to benefit from the augmentation. It is important to feed the network images that it actually will encounter in the real world. For the depth estimation task, some common and useful augmentations are; small random rotations, gaussian noise, random horizontal flips, color augmentations and translations.

**2.5.3 Error metrics**

To quantitatively evaluate depth predicting networks the error metrics used by Eigen *et al.* [15] are commonly used. Below the error metrics are defined where \(y_i\) denotes the prediction and \(y^*_i\) the ground truth value of pixel \(i\), \(T\) denotes the

\(^2\)https://github.com/mrharicot/monodepth/tree/master/utils
total number of valid pixels.

\[
RMSE = \sqrt{\frac{1}{T} \sum_i \|y_i - y_i^*\|^2} \tag{2.8}
\]

\[
RMSE(\log) = \sqrt{\frac{1}{T} \sum_i \|\log(y_i) - \log(y_i^*)\|^2} \tag{2.9}
\]

\[
\text{SILog} = \frac{1}{T} \sum_i d_i^2 - \frac{1}{T^2} \left( \sum_i d_i \right)^2, d_i = \log(y_i) - \log(y_i^*) \tag{2.10}
\]

The SILog error metric was introduced by Eigen et al. [15] to measure the relationship between points in the scene irrespective of the absolute global scale, thus all scalar multiple of \( y \) will give the same error. On KITTI the benchmarking is ranked according to the SILog error.

\[
\text{ARD} = \sqrt{\frac{1}{T} \sum_i \frac{|y_i - y_i^*|}{y_i^*}} \tag{2.11}
\]

\[
\text{SRD} = \sqrt{\frac{1}{T} \sum_i \frac{||y_i - y_i^*||^2}{y_i^*}} \tag{2.12}
\]

\[
\frac{1}{T} \sum_i \left( \max \left( \frac{y_i}{y_i^*}, \frac{y_i^*}{y_i} \right) = \delta < \text{thr} \right), \text{thr} = [\lambda, \lambda^2, \lambda^3] \tag{2.13}
\]

This error metric, known as accuracy with a threshold, divides the error ratios, \( \delta \), into intervals determined by the threshold value \( \lambda \). The accuracy (in percent) is the number of pixels with a error ratio less than the threshold, divided by the total number of pixels. Typically \( \lambda \) is 1.25, which corresponds to a prediction accuracy of \( \{\pm 25\%, \pm 56.25\%, \pm 95.31\%\} \) compared to the ground truth value.
In this chapter the implemented method is described. The method can be divided into two phases. The first phase involves retraining a pretrained network on a new dataset and evaluating the performance. The second phase involves modifying the network architecture with the aim to improve the performance on the depth estimation task.

3.1 Retraining a pretrained network

Since there already exist a large amount of depth estimating networks that have proven successful, one of these was used as starting point. The network chosen as starting point was the one proposed by Laina et al. [29], briefly described in section 2.5.

This network was chosen because it had performed well on the NYU-Depth V2 dataset [31] and had a comparable simple structure, which would facilitate future modifications. Also, no additional inputs or post-processing steps were used to train the network, in contrary to some of the other high performing networks [5, 23, 27]. Further, a PyTorch implementation of the network was found as open-source on GitHub\(^1\) along with pretrained weights on the NYU-Depth V2 dataset, which also was a motivation for choosing this network. From now on, this network [29] will be referred to as the baseline.

This network had only been trained on the indoor NYU-Depth V2 dataset [31] and no reliable benchmarking of the network was found on the KITTI dataset. Retraining and evaluating it on the KITTI dataset would provide an initial

---

\(^1\)https://github.com/XPFly1989/FCRN
sanity check of the network performance. The evaluation would also work as a decision basis, whether to modify the network for a performance boost or if a better approach would be to use another network architecture as starting point.

### 3.1.1 Network architecture

A brief description of the baseline is presented in section 2.5 and a more detailed description and the architecture can be found in the original paper [29]. A simplified illustration of the baseline architecture is shown in figure 3.1.

![Baseline architecture](image)

**Figure 3.1: Baseline architecture.**

The baseline architecture follows an encoder-decoder structure. The encoder is based on ResNet-50 and starts with convolution, batch normalization, a ReLU activation and a max-pooling layer. Next follows four ResNet blocks, each containing several convolutions and residual skips. More details about the ResNet architecture and the residual skips are found in [22]. After the ResNet blocks, convolution and batch normalization are applied to reduce the number of feature maps. The encoder outputs several feature maps with spatially lower resolution than the input image.

The first part of the decoder consists of four learnable upsampling blocks, returning an output that is approximately half the size of the input image. Then, a drop-out layer with a probability of 50%, convolution and a ReLU activation are applied. Finally the output is upsampled by bilinear interpolation, resulting in a final prediction with the same spatial size as the input. More details about the upsampling blocks are found in [29].

### 3.2 Modifying network architecture

The baseline network [29] was modified in various ways by adding, removing and replacing parts of the original architecture to improve its performance.
3.2 Modifying network architecture

3.2.1 Network architectures

Four different modifications were made and combined in different ways to find the combination giving the best performance. If one modification was found to improve the performance it was used in addition with the next modification. This was done since there was not enough time to try all possible combinations. The modifications are described in the sections below.

Incorporating low level feature maps to the decoder

Inspired by the encoder-decoder network, DeepLabv3+ proposed by Chen et al. [9], a set of low level feature maps from the encoder were added to the decoder to help with the upsampling. DeepLabv3+ is a network originally made for semantic segmentation, which is closely related to the task of depth estimation [29], thus what works well for semantic segmentation is likely to also work well for this task.

The low level feature maps, obtained as the output from the first ResNet block, were sent through an 1x1 convolution to reduce the number of channels, followed batch normalization and a ReLU activation. The low level feature maps were then upsampled using bilinear interpolation to match the spatial size of the feature maps obtained from the third upsampling block. This was necessary since the feature map sizes must match for concatenation to be possible. The feature maps were concatenated and sent as input to the final fourth upsampling block. The incorporation of the low level feature maps is shown in figure 3.2.

Figure 3.2: Modified network architecture incorporating low level feature maps from the encoder to the decoder. Feature maps obtained from the first ResNet block are concatenated with the feature maps obtained from the third upsampling block.
Incorporating an ASPP module

Another modification made to the network was also inspired by the works of Chen et al. [7–9]. An ASPP-module, first introduced in [7], was added to the encoder to help extract features and to capture multi-scale information. A similar ASPP-module was used by Fu et al. [16] in their state-of-the-art depth network DORN, to capture objects and image context at different scale.

The implementation of the ASPP-module follows the work in [8] by removing the stride in the last ResNet block and applying atrous convolution with a dilation rate of 2. The ASPP-module was added after the last ResNet block. The output stride, which is the ratio between the input image spatial resolution and the encoder output resolution [8], was set to 16. For the baseline encoder the output stride was 32, thus by incorporating the ASPP-module, the feature maps from the encoder obtained a spatial size twice as large and therefore one upsampling block could be removed. For the networks incorporating the ASPP-module, only three upsampling blocks were used. The addition of the ASPP-module is shown in figure 3.3.

![Modified network architecture incorporating the ASPP-module.](image)

The integrated ASPP-module contains a 1x1 convolution, three 3x3 atrous convolutions with dilation rates 6, 12 and 18 respectively and a global average pooling layer. The operations were performed in parallel and the resulting features from all branches were concatenated and sent through another 1x1 convolution, batch normalization and ReLU activation. This is shown in figure 3.4.

Figure 3.3: Modified network architecture incorporating the ASPP-module. The ASPP-module is added after the last ResNet block. Note that only three upsampling blocks are used when adding the ASPP-module.
3.2 Modifying network architecture

Figure 3.4: The ASPP-module. Convolutions with different dilation rates and global average pooling are performed in parallel. The resulting feature maps are concatenated and sent through another convolution, batch normalization and ReLU activation.

**Replacing ReLU with PReLU**

He et al. [21] introduced PReLU (discussed in section 2.2.2) and show that using PReLU instead of ReLU leads to faster convergence and lower errors for both training and validation. Further, it improves model fitting while still having a small risk of overfitting to the data. They made their experiment on the ImageNet 2012 classification dataset and their results were the first to outperform humans on this task. Even though He et al. [21] use PReLU for a classification task, there are reasons to believe that it would work similar for the depth estimation task since they both are image related problems.

With this motivation, all ReLU activation functions were replaced with PReLU, except for the first two blocks of the ResNet backbone that were frozen, as will be explained in section 3.3.4. Since He et al. [21] conclude that channel-wise and channel-shared PReLU perform comparably, the channel-shared version was chosen for simplicity and to keep the number of parameters as low as possible.

**Changing feature extractor to DenseNet**

In [29] ResNet-50 is used as encoder to extract features from the input image. ResNet has shown great potential by applying skip connections that allows for steps to be skipped, which ease the training of deep networks [22]. In some of the works [5, 16, 29] described in section 2.5, different architectures have been evaluated against each other with ResNet outperforming all others. According to Cao et al. [5] and Eigen et al. [14] it also seems like deeper architectures perform slightly better than shallow ones when predicting depth.

Even though ResNet has shown great performance, there exist a variety of other networks that can be used to extract features and that might work better for this particular task. DenseNet, proposed by Huang et al. [24], is a very deep but narrow network in which all layers are connected directly to each other. In DenseNet, each layer receives inputs from all previous layers and passes its own feature maps to all future layers. This ensures maximum information flow between layers and ease the training. In contrary to ResNet, in which the features are summed together, the feature maps in DenseNet are concatenated.
More details about the DenseNet architecture are found in [24].

Since DenseNet improves the flow of information, strengthens the feature propagation through the network, and uses a type of skip-connections, similar to the ones that have proven useful in ResNet, DenseNet was assumed to have potential as a good feature extractor. At the same time, using DenseNet instead of ResNet would reduce the number of parameters needed. Therefore, the ResNet backbone was replaced with DenseNet-201.

The implementation of DenseNet-201 follows [24], but the last global average pooling layer and the fully connected layer were removed. Instead an 1x1 convolution and batch normalization were applied to reduce the number of channels before sending the feature maps into the decoder.

3.3 Network training

This section describes how the network training was performed. The implementation, training and evaluation was done in Python using the PyTorch backend. The networks were trained on a NVIDIA GeForce RTX 2080 GPU with 8GB memory.

3.3.1 Datasets

The public KITTI dataset [42] [18], described in section 2.5.2, was used for training and evaluation. The dataset was split according to the split proposed by Eigen et al. [15] presented in section 2.5.2. This split was used to train all network architectures and allowed for a fair comparison between the networks.

The synced and rectified RGB images, from both the left and right camera, were used as input to the network. The depth maps provided by Uhrig et al. [42] were used as ground truth since these were denser than the raw LiDAR scans. After downloading the KITTI dataset, it was found that the corresponding depth maps for some RGB images were missing. These images were removed from the dataset, making the dataset slightly smaller, although this was assumed to not have any larger effect on the training. The exact number of images used in each set is shown in table 3.1.

<table>
<thead>
<tr>
<th>Set</th>
<th>Number of images</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train</td>
<td>44 492</td>
</tr>
<tr>
<td>Validation</td>
<td>1 744</td>
</tr>
<tr>
<td>Test</td>
<td>1 304</td>
</tr>
</tbody>
</table>

*Table 3.1: The total number of images in each set.*
3.3 Network training

For evaluating the network ability to generalize to new scenes the CityScapes dataset [11] was used. The CityScapes is a large scale dataset containing stereo video sequences recorded in 50 different cities. The dataset was created for semantic urban scene understanding and does not provide ground truth depth. Therefore, the evaluation was only done visually.

3.3.2 Data preprocessing and augmentation

Before sending the images and ground truths into the network, they were preprocessed. The full resolution images were first subsampled to size 350x270 using nearest neighbour and a random crop, either of size 304x228 or 304x240 depending on the network architecture, was extracted. This was made to reduced the training time while still providing the network with images containing a full view of the scene, something that Eigen et al. [14] highlight the importance of when predicting depth. If taking the same crop but from the full resolution image instead, each patch would contain less global context, making it harder for the network to learn from monocular cues.

The baseline network was originally implemented to receive inputs of size 304x228 and therefore this size was used initially. However, it was later changed to 304x240 to ease the incorporation of the low level feature maps. By changing the input size, the low level feature maps to be concatenated automatically obtained the same spatial size, thus removing the need of interpolation.

The ground truth images were converted to float images and each pixel value was divided by 256 to obtain the depth in meters. Thus, the network was trained on ground truths in meters and consequently also outputs predictions in meters.

During the training and validation phase, data augmentation was used. From the conclusions drawn in section 2.5.2 regarding data augmentation, horizontal flipping with a chance of 50% and random rotation of $\pm 5^\circ$ were applied to the images. This was made after the subsampling but before extracting the random crop. Further on, a small amount of gaussian noise was applied.

3.3.3 Mask function

To handle the sparsity of the depth maps and to not let it interfere with the training, a binary mask was constructed for each depth map. The mask contained ones where depth values were valid and zeros where there were no valid depth values. The mask was multiplied with the predicted image in the loss function to extract only the pixels with a corresponding ground truth, then the loss was calculated only over these pixels. The mask was also used during the evaluation to mask out the invalid pixels.
3.3.4 Weight initialization and frozen layers

The baseline network was initialized with pretrained weights from the NYU-Depth V2 dataset to make the network converge faster. To further reduce training time, the parameters of all layers up to the second ResNet block were frozen. This was motivated by the fact that the early layers of CNNs extract low level features, which were assumed to be similar for the two datasets.

The modified networks were also initialized with these pretrained weights for the parts that remained the same. The new parts, not having any pretrained weights, were initialized using the Kaiming initialization [21]. The Kaiming weight initialization has proven particularly suited when using non-linear activation functions such as ReLU or PReLU [21].

For the modified networks using ResNet as feature extractor, the first layers were frozen in the same way as for the baseline. For the network using DenseNet as feature extractor, no pretrained weights from any depth dataset were available to initialize the encoder with. For that reason it was initialized using the Kaiming method [21]. Since these weights are random, they most likely are far from optimal and it was considered essential to train all layers, meaning updating all parameters, for the network to learn properly. Thus, for the network with DenseNet no layers were frozen.

3.3.5 Optimization

During the optimization, the loss function is minimized with respect to the parameters as described in section 2.3.2. The parameters that the user can decide, such as the learning rate, batch size, number of epochs etc., are referred to as the hyperparameters and can have a large effect on the result of the training. This section describes how the hyperparameters were chosen.

To train the baseline and all network variants, the ADAM optimizer was used due to its compelling advantages presented in section 2.3.2. The same learning rate, found by trial and error, was used to train all networks and it was lowered after a fixed number of epochs. At which epoch to lower the learning rate and the total number of epochs to train the networks, were found by studying the loss of the baseline after training it for a large number of epochs.

According to Goodfellow [20], using a large batch size is desirable since it provides a more accurate estimate of the gradient. Therefore, the largest possible batch size, which was limited by the GPU memory, was used during training.

The BerHu loss was used as loss function since it incorporates the positive properties of both the $L_1$ and $L_2$ losses, as described in section 2.3.1. Laina et al. [29] and He et al. [23] yielded more accurate depth estimations using BerHu over $L_2$ and $L_1$ loss respectively, which also motivated for using this loss function.
For the network architecture that was found to perform the best, some training changes were made to find out if the performance could improve further.

### 3.4 Evaluation

The model that gave the lowest loss on the validation set was picked out and used for evaluation on the test set. The evaluation was done quantitatively, using the error metrics in section 2.5.3, and qualitatively using visual inspection. The average of each error metric was calculated over the entire test set.

During evaluation the images were subsampled in the same way as during the training phase, although no data augmentation was applied. A center crop was extracted to make the network always perform the evaluation on the same images. Since the network was fed subsampled and cropped images with smaller spatial size than the originals, the network also output predictions of these sizes. The raw outputs, without any post-processing, were compared to the subsampled and cropped ground truths to evaluate the network performance. Consequently, the evaluation was done at a lower resolution than the original. Only the pixels with valid corresponding ground truth values were used in the evaluation and the binary mask, described in section 3.3.3, was used to mask out the invalid pixels.

For image related problems, a visual inspection is always important. As Eigen et al. [15] discovered, a visual difference can be noticeable even if the error metrics do not differ significantly. There are no standard methods for measuring the visual performance of depth maps. However it seemed reasonable to take particular consideration to boundaries and shapes of different objects, as well as the ability of the network to find details in the scene. The networks that were able to do this were considered superior.

The best performing network architecture together with the best training settings was used to perform predictions on the Cityscapes dataset [11]. By only training the network on KITTI but evaluating it on CityScapes, the network ability to generalize to new scenes was evaluated. Since the Cityscapes dataset does not provide ground truth depth, only a visual inspection was possible.
In this chapter, the results of the implemented method are presented. First, the modified network architectures and chosen training settings are stated followed by the quantitative and qualitative results.

4.1 Modified network architectures

Apart for the baseline, six different network architectures were implemented. These are presented in table 4.1 that states which modifications that were made to each network. The different modifications are described in section 3.2.1.

<table>
<thead>
<tr>
<th>Network name</th>
<th>Feature extractor</th>
<th>ASPP</th>
<th>Activation func.</th>
<th>LL feature maps</th>
<th>Upsampling blocks</th>
<th># parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>ResNet-50</td>
<td></td>
<td>ReLU</td>
<td>0</td>
<td>4</td>
<td>62.127 M</td>
</tr>
<tr>
<td>Mod-LL</td>
<td>ResNet-50</td>
<td></td>
<td>ReLU</td>
<td>1</td>
<td>4</td>
<td>62.293 M</td>
</tr>
<tr>
<td>Mod-LL.p</td>
<td>ResNet-50</td>
<td></td>
<td>PReLU</td>
<td>1</td>
<td>4</td>
<td>62.293 M</td>
</tr>
<tr>
<td>Mod-ASPP-LL</td>
<td>ResNet-50</td>
<td>x</td>
<td>ReLU</td>
<td>1</td>
<td>3</td>
<td>40.189 M</td>
</tr>
<tr>
<td>Mod-ASPP-LL2</td>
<td>ResNet-50</td>
<td>x</td>
<td>ReLU</td>
<td>2</td>
<td>3</td>
<td>41.516 M</td>
</tr>
<tr>
<td>Mod-DenseNet</td>
<td>DenseNet-201</td>
<td></td>
<td>ReLU</td>
<td>0</td>
<td>4</td>
<td>58.025 M</td>
</tr>
<tr>
<td>Mod-ASPP-LL.p</td>
<td>ResNet-50</td>
<td>x</td>
<td>PReLU</td>
<td>1</td>
<td>3</td>
<td>40.189 M</td>
</tr>
</tbody>
</table>

Table 4.1: Overview of the modified network architectures and which modifications that were made to each of them. The table states the network name, the feature extractor, whether an ASPP-module was incorporated or not, the activation function, the number of sets of low level (LL) feature maps incorporated from the encoder to the decoder and the number of upsampling blocks. Also, the number of total parameters to update for each network is shown.
The networks incorporating the ASPP module only contain three upsampling blocks, thus when adding the low level feature maps for these networks, they were added to the second upsampling block and not the third as described in section 3.2.1. For Mod-ASPP-LL2, two sets of low level feature maps were added to the decoder. The first one from the first ResNet block to the second upsampling block, and the second set from the second ResNet block to the first upsampling block.

### 4.2 Training settings

The baseline network was initially trained for 120 epochs with a learning rate of $10^{-4}$. The learning rate was lowered to $10^{-5}$ after 55 epochs when the validation loss had reached a plateau. From epoch 55 to epoch 120, there was only a tiny decrease of the validation loss and this insignificant gain in performance was assumed to not be justified by the extra training time. Therefore, the choice was made to only train the networks for 60 epochs, which was enough for the networks to converge and give good results.

The modified networks were trained for 60 epochs with a learning rate of $10^{-4}$ which was lowered to $10^{-5}$ after 55 epochs. The largest possible batch size was found to vary slightly between the networks and for most networks a batch size of 16 was used. However, for some networks, 12 was the largest possible batch size. The training settings and input image size of each network are specified in table 4.2.

<table>
<thead>
<tr>
<th>Network</th>
<th>Learning rate</th>
<th>Batch size</th>
<th>Learning rate lowered at epoch</th>
<th>Input size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>$10^{-4}$</td>
<td>16</td>
<td>55</td>
<td>304x228</td>
</tr>
<tr>
<td>Mod-LL</td>
<td>$10^{-4}$</td>
<td>16</td>
<td>55</td>
<td>304x240</td>
</tr>
<tr>
<td>Mod-LLp</td>
<td>$10^{-4}$</td>
<td>12</td>
<td>55</td>
<td>304x228</td>
</tr>
<tr>
<td>Mod-ASPP-LL</td>
<td>$10^{-4}$</td>
<td>16</td>
<td>55</td>
<td>304x240</td>
</tr>
<tr>
<td>Mod-ASPP-LL2</td>
<td>$10^{-4}$</td>
<td>16</td>
<td>55</td>
<td>304x240</td>
</tr>
<tr>
<td>Mod-denseNet</td>
<td>$10^{-4}$</td>
<td>12</td>
<td>55</td>
<td>304x228</td>
</tr>
<tr>
<td>Mod-ASPP-LLp</td>
<td>$10^{-4}$</td>
<td>16</td>
<td>55</td>
<td>304x240</td>
</tr>
</tbody>
</table>

Table 4.2: Overview of the training of the different network architectures. The table states the learning rate, batch size, at which epoch the learning rate was lowered a factor 10 and the input (and consequently also the output) size.

The changed training settings conducted on the best performing network are
presented in table 4.3. When changing to the full resolution images as input, the batch size was set to four, which was the largest possible. The training time for each epoch increased significantly and due to time limitations the network was only trained for 15 epochs. However, this accounted for the same number of weight updates as when using batch size 16 and training for 60 epochs. The learning rate was set to $10^{-4}$.

<table>
<thead>
<tr>
<th>Training setting nr</th>
<th>Loss func.</th>
<th>Learning rate</th>
<th>Batch size</th>
<th>Input size</th>
<th>Total epochs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BerHu</td>
<td>$10^{-4}$</td>
<td>4</td>
<td>1200x352</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>$\ell_1$</td>
<td>$10^{-4}$</td>
<td>4</td>
<td>1200x352</td>
<td>15</td>
</tr>
</tbody>
</table>

*Table 4.3*: Overview of the changed training settings conducted on network Mod-ASPP-LLp.

### 4.3 Quantitative results

In this section, the quantitative results are presented. First the results of the modified network architectures are presented, followed by the results of the changed training settings conducted on Mod-ASPP-LLp, which was found as the best performing network architecture.

#### 4.3.1 Network architectures

Here, the validation loss and the results of the error metrics of the network architectures stated in 4.1 are presented. The networks were trained on KITTI dataset with Eigen split [15] using the training settings in table 4.2.

The validation loss is shown in figure 4.1. It can be noticed that, at epoch 55, when the learning rate was lowered, the loss drops significantly for all networks. It was found that, even if lowering the learning rate further, the loss did not drop again.
Figure 4.1: The validation loss of the different network architectures presented in table 4.1 as a function of the number of epochs.

Higher is better
Lower is better

<table>
<thead>
<tr>
<th>Network</th>
<th>δ &lt; 1.25</th>
<th>δ &lt; 1.25²</th>
<th>δ &lt; 1.25³</th>
<th>ARD</th>
<th>SRD</th>
<th>RMSE_{lin}</th>
<th>RMSE_{log}</th>
<th>SILog</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>0.869</td>
<td>0.965</td>
<td>0.990</td>
<td>0.112</td>
<td>0.647</td>
<td>4.273</td>
<td>0.165</td>
<td>15.128</td>
<td>17.4h</td>
</tr>
<tr>
<td>Mod-LL</td>
<td>0.868</td>
<td>0.965</td>
<td>0.990</td>
<td>0.112</td>
<td>0.640</td>
<td>4.247</td>
<td>0.165</td>
<td>15.030</td>
<td>19.2h</td>
</tr>
<tr>
<td>Mod-LLP</td>
<td>0.884</td>
<td>0.973</td>
<td>0.992</td>
<td>0.101</td>
<td>0.595</td>
<td>4.260</td>
<td>0.156</td>
<td>14.256</td>
<td>22.5h</td>
</tr>
<tr>
<td>Mod-ASPP-LL</td>
<td>0.874</td>
<td>0.966</td>
<td>0.990</td>
<td>0.110</td>
<td>0.623</td>
<td>4.184</td>
<td>0.163</td>
<td>14.892</td>
<td>21.3h</td>
</tr>
<tr>
<td>Mod-ASPP-LL2</td>
<td>0.875</td>
<td>0.967</td>
<td>0.991</td>
<td>0.109</td>
<td>0.640</td>
<td>4.225</td>
<td>0.162</td>
<td>14.806</td>
<td>26.5h</td>
</tr>
<tr>
<td>Mod-DenseNet</td>
<td>0.858</td>
<td>0.962</td>
<td>0.988</td>
<td>0.120</td>
<td>0.698</td>
<td>4.409</td>
<td>0.173</td>
<td>15.859</td>
<td>27.7h</td>
</tr>
<tr>
<td>Mod-ASPP-LLP</td>
<td>0.892</td>
<td>0.974</td>
<td>0.993</td>
<td>0.099</td>
<td>0.581</td>
<td>4.123</td>
<td>0.151</td>
<td>13.846</td>
<td>22.7h</td>
</tr>
</tbody>
</table>

Table 4.4: Quantitative results of the network architecture on the KITTI dataset, using the split proposed by Eigen et al. [15]. The table shows the average error metrics over the entire test set. The error metrics were calculated in meters at the input image resolution and only the pixels with valid corresponding ground truths were taken into account. The best values are marked in bold and the next best in cursive.

Table 4.4 presents the average results of the error metrics calculated over the entire test dataset. The error metrics were calculated in meters at the input image resolution and only the pixels with valid corresponding ground truths
were taken into account. Also the training time for each network is shown in the table.

Due to time limitations it was not possible to perform the changed training settings for all networks. Therefore the network that was considered having the best performance was chosen for this. By studying table 4.4 it is clear that Mod-ASPP-LLp achieves the best result on all error metrics, therefore the changed training settings were only conducted on this network.

### 4.3.2 Changed training settings

Table 4.5 presents the results of the error metrics of Mod-ASPP-LLp when trained with the different training settings. Note that the evaluation was performed at the input image resolution. When using the standard training settings, the evaluation was made at resolution 304x240 but for training setting 1 and 2 it was performed at resolution 1200x352. This was done to avoid introducing interpolation errors that would affect the evaluation.

<table>
<thead>
<tr>
<th>Training settings</th>
<th>( \delta &lt; 1.25 )</th>
<th>( \delta &lt; 1.25^* )</th>
<th>ARD</th>
<th>SRD</th>
<th>RMSE(_{lin} )</th>
<th>RMSE(_{log} )</th>
<th>SIlog</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard</td>
<td>0.892</td>
<td>0.974</td>
<td>0.993</td>
<td>0.099</td>
<td>0.581</td>
<td>4.123</td>
<td>0.151</td>
<td>13.846</td>
</tr>
<tr>
<td>1</td>
<td>0.908</td>
<td>0.983</td>
<td>0.996</td>
<td>0.089</td>
<td>0.430</td>
<td>3.474</td>
<td>0.134</td>
<td>12.498</td>
</tr>
<tr>
<td>2</td>
<td>0.913</td>
<td>0.983</td>
<td>0.995</td>
<td>0.085</td>
<td>0.428</td>
<td>3.430</td>
<td>0.132</td>
<td>12.293</td>
</tr>
</tbody>
</table>

**Table 4.5:** The average error metrics over the entire test dataset for network Mod-ASPP-LLp with changed training settings according to table 4.3. The error metrics were calculated in meters at the input image resolution, which differed between the training settings. Only the pixels with valid corresponding ground truths were taken into account. The best values are marked in bold.

It is clear that when training was performed on the full resolution images the error metrics improve remarkably. Network Mod-ASPP-LLp with training setting 2 will from now on be referred to as DepNet and is considered the best performing model.

The error metrics of DepNet is in table 4.6 compared to the results of the methods proposed by Eigen *et al.* [15]\(^1\), Fu *et al.* [16]\(^2\), Godard *et al.* [19]\(^3\), Garg *et al.* [17] and Cao *et al.* [5]. These methods had all been trained on the Eigen split however using the sparser LiDAR scans as ground truths.

---

\(^1\) Train on sparser ground truth and evaluation done at 1/4 of original resolution
\(^2\) Train on sparser ground truth.
\(^3\) Train on sparser ground truths and evaluation done at original resolution.
4.4 Qualitative results

The predicted depth maps of the networks, trained according to table 4.2 on the KITTI dataset using Eigen split [15], are presented in figure 4.2 - 4.6. In figure 4.7 the qualitative results for Mod-ASPP-LLp trained with the training settings in table 4.3 are presented. The images, all from different scenes, were chosen for the visual inspection since they highlight different strengths and weaknesses of the networks. All images were taken from the test set to evaluate the ability of the networks to generalize to data they have not seen before.

All predictions, except for when training settings 1 and 2 were used, have been interpolated by bilinear interpolation to the presented size. This was done to get a better visualization and to make a visual inspection easier. Note that, because the maximum predicted depth value differs between networks, each prediction has its own color scale, presented by the color bar on the right hand side. The color bar shows the predicted depth in meters, where red corresponds to large depth values and blue to small. In each figure, the input image, the corresponding depth ground truth and the prediction of each network are shown. The images are best visualized on a digital medium.
Figure 4.2: The output predictions of the different network architectures on test image number 1. Note how network (c) and (h) more of less fail to detect the traffic sign to the right. Network (f), (g) and (i) more accurately predict the sky.
Figure 4.3: The output predictions of the different network architectures on test image number 2. All networks struggle to find and separate the cars in the image.
4.4 Qualitative results

Figure 4.4: Qualitative results of the network architectures on test image number 3. Note how network (d)-(g) and (i) manage to find the round shape of the traffic sign to the left better than network (c) and (h).
Figure 4.5: Qualitative results of the network architectures on test image number 4. All networks struggle with finding the correct shape of the truck and the trees to the right. Network (f)-(i) manage to more accurately predict the top part of the image.
Figure 4.6: Qualitative results of the network architectures on test image number 5. All networks fail to detect the person in front of the car as well as the person in the bottom left corner.
Figure 4.7: The output predictions of Mod-ASPP-LLp with different training settings. Note the sharper boundaries and the more accurate shapes in (d) and (e) compared to (c).
4.5 Generalization to CityScapes dataset

Here, the visual results of DepNet on five images from the CityScapes dataset are presented. The CityScapes images, having a spatial size of 2048x1024, were center cropped to 1200x352 before they were sent into the network. Note that the network had only been trained on the KITTI dataset and not the CityScapes dataset.

Figure 4.8: Qualitative results of the best performing network, DepNet, on the CityScapes dataset [11]. Note how the network has difficulties with reflections in car windows and close objects.
In this chapter the method presented in chapter 3 and the results from chapter 4 are discussed and analyzed. Also future work is included in this chapter.

5.1 Discussion

The main goal of master thesis; to implement, train, and evaluate a neural network that predicts depth from monocular RGB images, has been achieved. The best performing network, DepNet, manages to predict the depth from monocular images with results comparable to state-of-the-art methods.

5.1.1 Method

In this section the implemented method is discussed.

Dataset limitations

The performance of a network is strongly related to the quality and size of the dataset it is trained on. Since the proposed network, DepNet, is fairly complex and contains a large number of parameters (approximately 40 millions), a large dataset is required to avoid overfitting. It is possible that the approximately 45000 images used to train the network were not sufficient. Even if the network showed promising results on KITTI, it became clear that the network had a generalization problem when predicting the depth from the CityScapes dataset. To improve its ability to generalize, the network could be trained on a larger amount and more varied data.

Another important property of the dataset is its accuracy of the ground truths.
Even if the denser depth maps provided by Uhrig et al. [42] were considered accurate and are made public for depth estimation tasks, it is possible that they contain some faulty values. Also, the fact that the LiDAR scan suffers from the rolling shutter problem, can mean that the raw LiDAR scans that were used to obtain the denser depth maps, were not completely accurate to begin with. Exactly how this affects the network performance is unclear but it is worth to keep in consideration.

Since the ground truth data was sparse and did not contain values in all pixels, the network had to learn to fill in the missing values. For areas that never were given any depth information, the network had a hard time predicting accurate values for. This is in particular reflected in the images shown in section 4.4, where the upper parts, which no ground truth values were given for, are predicted poorly.

By looking at the validation loss of the network architectures in figure 4.1 and the error metrics in table 4.4, it is noticed that these two do not match. In general, the validation set is used to pick the model that generalize the best, meaning the model that is most likely to perform well on the test set. However, this does not seem to be the case here. For example, Mod-DenseNet has lower validation loss than the baseline and Mod-LL, but still gives worse results on the test set. This is most likely a sign of that the validation set is too similar to the training set and too different from the test set. Since the validation set does not represent the distribution of the test set, it is difficult to pick out the best generalizing model since possible overfitting will not be detected. To overcome this problem, another split of the dataset or another dataset could be used. However, due to time limitations this was not performed during this thesis work.

**Evaluation analysis**

To evaluate the performance of the networks, both the visual results and the error metrics were studied. The error metrics give an objective assessment of the average performance over the entire test set, while the visual inspection, which only was conducted over five images of a total of 1304, can not be considered to present the whole dataset distribution. Also, a visual inspection is always more or less subjective. To obtain a more trustworthy qualitative evaluation, a larger amount of predictions should be inspected and evaluated by several persons in a more structured way. Because of this, the quantitative results were considered as more reliable and larger weight was put on these when picking out the best network architecture.

However, the quantitative evaluation also have limitations. One important thing to keep in mind is that the error metrics only measure how well the network manages to predict the depth values relative the provided ground truth. This means that the network can predict completely wrong depth values in a lot of pixels and still achieve good quantitative results, as long as it predicts the
correct depth of the pixels having corresponding ground truths. This is a disadvantage when working with sparse data and because of this, a visual inspection was still considered important.

In table 4.4 the error metrics are compared to some state-of-the-art methods. These method had been trained on the sparse LiDAR scans and not the denser depth maps provided by Uhrig et al.[42], which were used to train the proposed network. The evaluation was also made at different resolutions. For example, Eigen et al. [15] performed their evaluation at 1/4 of the original resolution while Godard et al. [19] interpolated their predictions to the original size before evaluation. Also, it has to be kept in consideration that DepNet was trained on a slightly reduced split of the dataset. Even if this was considered insignificant in terms of the training, it might have had some affect on the evaluation since a number of images were missing from the test set. Due to these difference, the comparison between DepNet and the other methods are not completely fair, yet it gives an indication of the relative performance and potential of the network.

Training procedure

The search of hyper-parameters was not made to a great extent. Some hyper-parameters, for example the learning rate and batch size, were found by trial and error, while other such as the optimizer and weight decay, never were varied. Further, the performed hyper-parameter search was only done for the baseline network, then the same settings were used to train the modified networks. With more time, more effort could have been put into finding the optimal hyper-parameters for each network architecture and obtaining a more fair idea about their true potential.

To train the different networks the BerHu loss function was used, which puts large weight on large prediction errors. By punishing large errors harder, the resulting RMSE will most likely be lower than if the same weight was put on all errors. Since the RMSE is a common error metric when predicting depth, it is of course desirable to keep it as low as possible. This could be the reason why BerHu and $L_2$ loss are popular loss functions for this task. However, in some applications, for example autonomous driving, it could be sensible to allow larger errors for larger depths and prioritize accurate depth estimates of close objects. For example, if the network predicts 70m instead of 65m it might not matter as much as if it predicts 6m instead of 1m. By using another loss function that puts more weight on smaller error, the closer predictions could be estimated more accurately. Of course, the application of the depth predictions has to be kept in mind, for some applications accurate depth predictions of close objects is more important than for objects farther away, while for other applications it is the opposite.
Network architecture

The proposed network, DepNet, is a relatively large and complex network containing over 40 millions parameters. Even if it has less parameters than the baseline, the fact that the model trains on high resolution images puts a high demand on computational and memory resources. The high complexity of the network also limits the possibilities of real-time performance.

One benefit with the network is that it is not limited to a fixed input size. The network can take in any image with a size divisible by 16 (assuming the GPU memory is sufficient) and return the output. Further, it could easily be modified to take in an arbitrary sized image, however this would involve some interpolation steps. DepNet can also easily be adapted to the task of semantic segmentation or other pixel-wise dense prediction tasks. This property in combination with the non-fixed input size, makes it easy to retrain the network for new tasks using different datasets and input sizes.

5.1.2 Results

In this section the results presented in chapter 4 are analyzed and discussed.

Qualitative results on KITTI dataset

Through visual inspection of the predictions shown in figure 4.2-4.6, various differences between the networks can be noticed.

In general, the baseline and Mod-DenseNet make smoother predictions and tend to blend adjacent objects together, while the other networks show sharper boundaries, detect more correct shapes, and are slightly better at separating objects. This is particular observed by looking at the shape of the traffic signs in figure 4.2 and 4.4. However, the baseline and Mod-DenseNet seem better at predicting continuous shapes and finding the full height of posts and trees. The other networks tend to make somewhat deformed and discontinuous predictions, something that is particular noticable in figure 4.4 and 4.5. In figure 4.2, 4.4 and 4.5 it is clear that the networks using the ASPP-module manage to predict the top part of the images more accurately compared to the networks not containing the ASPP-module. A strange behaviour can be seen for Mod-ASPP-LLp that predicts a hollow ellipse at the top of all images. This is not the case for any of the other networks.

In summary, all networks manage to find large objects in the scene but struggle with finding small objects and the correct shape of them. This could be because some high resolution information is lost due to the subsampling process. Although, the networks incorporating low level feature maps succeed in finding small objects and also predict sharper boundaries better than the other networks. This is probably because the decoder is provided with low level but high frequency information from the encoder, which helps it to recover the
5.1 Discussion

Details. That the incorporation of the ASPP module improved the prediction of the top part of the images can be due to the enlarge receptive field that was gained when introducing atrous convolution.

The strange behavior of Mod-ASPP-LLp, that predicts a hollow ellipse at the top of all image, is somewhat confusing but likely related to the fact that no ground truth is ever provided for these parts. An easy experiment where the sky are given a ground truth value, can be conducted to see if the behavior disappears.

A large visual difference between the networks trained on subsampled images and the ones trained on the full resolution images can be seen in figure 4.7. Apart from the obviously significant better resolution and sharper boundaries, using the full resolution images also resulted in more continuous and correct shape of objects. The network became clearly better at finding small details, such as thin trees, lamp posts and traffic signs. This was expected, since if the network is provided with more details it will consequently also learn to find more details. Although, there might be some redundant information in the high resolution images and a good trade-off could be to downsample the images to half the size and use those for training. This could give similar performance but reduce the training time significantly.

**Qualitative results on CityScapes dataset**

By studying the predictions made on the CityScapes dataset in figure 4.8, it is clear that the network has a generalizing problem. In most cases it manages to make fairly good predictions and finds the most objects, but struggles with finding the correct boundaries. It also shows problems with reflections and predicting depth of close objects. However, these difficulties are not surprising as the two datasets, KITTI and CityScapes, have various differences. The images from CityScapes have a different coloring and brightness (the images are more dim) and another resolution. Further, it seems like the camera is positioned lower in CityScapes than in KITTI and the images are most likely captured using a camera with different camera parameters. All these differences cause trouble for the network since it has not seen these type of variations before and does not know how to handle them. On the other hand, the datasets also have many similarities. Both datasets were recorded in urban environments and contain similar objects such as cars, traffic signs, roads, people and buildings. This is probably the reason why the network still manages to make decent predictions.

Another possible reason for the generalization problem is that the network has overtrained and become to adapted to the KITTI dataset. If the network could be trained on more varied data, for example if color and brightness augmentation were applied together with random scaling, there are reasons to believe it would be able to generalize better to these scenes.
Quantitative results

Table 4.4 shows that all modified networks, except for Mod-DenseNet, perform better or equal to, than baseline on most error metrics. The incorporation of the low level feature maps, improved the RMSE, SRD and SILog, while the other error metrics more or less stayed the same as for the baseline. The networks using PReLU performed better than their ReLU variant on all error metrics, except for the RMSE for Mod-LLp, which was slightly worse than for Mod-LL. However, a significant decrease of the SILog error is obtained when switching from ReLU to PReLU, which is considered more important than the slightly increased RMSE.

By comparing Mod-LL and Mod-ASPP-LL it is noticed that the addition of the ASPP-module improved all error metrics, except for $\delta < 1.25^3$, which stayed the same. The most significant difference was seen for the RMSE that decreased from 4.247 to 4.184 when adding the ASPP-module. The comparison between Mod-ASPP-LL and Mod-ASPP-LL2 is not as straightforward, for most error metrics they show equal results, but Mod-ASPP-LL has lower RMSE and SRD while Mod-ASPP-LL2 has lower SILog.

Mod-DenseNet shows the worse quantitative results, which is unexpected since DenseNet was assumed to improve the feature extraction. One possible explanation for this is that, since no layers were frozen, the network was more prone to overfitting on the training data than the other networks were. It is also possible that the used hyper-parameters were not appropriate for training Mod-DenseNet, which in turn could have affected the results, however this was not investigated further.

Best performing network and training settings

Through the analysis of the qualitative and quantitative results in the sections above, the best performing network architecture was found to be Mod-ASPP-LLp. This network shows overall good visual results, are comparable good at predicting correct shapes and also manages to predict the sky relatively good. Further on, it outperforms the other networks on all error metrics as can be seen in table 4.4. Since the benchmarking on KITTI is ranked according to the SILog this was assumed to be the error metric of highest importance and Mod-ASPP-LLp obtained the lowest SILog with a good margin. Due to these reasons, Mod-ASPP-LLp was considered as the superior network architecture with the highest potential.

When finding the best training settings of the ones conducted, only the quantitative error metrics were taking into account since these were considered as more reliable compared to a subjective visual inspection. From table 4.5 it is clear that training setting 2, when training on the full resolution images with $\mathcal{L}_1$ loss, gave the best performance.
5.2 Future work

The fact that using the $\mathcal{L}_1$ loss instead of BerHu gave lower error metrics is unexpected. According to the reasoning in section 5.1.1 regarding the loss function, a loss function punishing large errors harder most likely should result in lower RMSE. One explanation to that this was not the case here, could be that the network employing $\mathcal{L}_1$ has faster convergence speed and obtains better results faster than when using the BerHu. However, further experiments are necessary to confirm this.

**Application in mono-SLAM**

The task of integrating the predicted depth maps into a mono-SLAM system pipeline was not performed in this thesis. However, several works, in particular [41] and [43], show that it is possible and that these systems outperform earlier monocular SLAM methods and show results comparable to stereo SLAM.

Since it is extremely hard implementing a network that make accurate predictions in all environments, the area of use has to be kept in mind in order to train the network on appropriate data. For example, DepNet, which had only been trained on an outdoor dataset would probably perform poorly if it was to be used indoors. By knowing in which environment the mono-SLAM system will be used, the network could be retrained to adapt better to this particular environment. Thus, before integrating the depth maps into the mono-SLAM system, the network might need to be retrained on another dataset and also a fine-tuning of the hyper-parameters should be made. However, these improvements are left for the future.

### 5.2 Future work

Apart from the future work of integrating the predicted depth maps into a mono-SLAM system as mentioned above, this sections presents three future work suggestions that focus on improving the network performance.

**Larger amount and more varied training data**

The network could be trained on more than just one dataset to increase the total dataset size. Further data augmentations such as spatial scaling, color, brighness and contrast augmentations could be applied to provide the network with more varied data. By doing this, the network would hopefully learn to generalize better to new scenes.

**Incorporate semantic information**

The network could be adapted to perform the task of semantic segmentation and depth estimation simultaneously. Since semantic labels and depth share context information, the semantic labels could be used to guide, and presumably improve, the depth prediction.
A more efficient decoder

This work has focused on the encoder part of the network architecture but the decoder also plays a significant role for the network performance. The decoder used in DepNet (first introduced in [29]) is rather complex and contains a lot of parameters. An idea is to replace it with a more efficient decoder to make the network more light-weighted and increasing the chance of future deployment in real-time applications.
In this thesis, the ill posed problem of monocular depth estimation has been investigated. To solve this problem, this thesis introduces a novel deep convolutional neural network, DepNet, that predicts depth from monocular RGB images by deploying an encoder-decoder structure. The proposed network architecture is inspired by state-of-the-art depth estimating networks. By incorporating dilated convolutions to capture context at different scales, skip-connections to recover high level details and PReLU activations to improve model fitting, the network achieves results comparable to state-of-the-art methods on the KITTI dataset.

The network can easily be adapted to other tasks, such as semantic segmentation, which could be used in combination with the depth estimation and possibly improve the performance. The network manages to estimate the depth of general objects in the scene and outputs high resolution depth maps, however it shows difficulties generalizing to new scenes. This concludes that further development, such as hyper-parameter tuning and more varied training data, are necessary before integrating it in a mono-SLAM system. Yet, the results are promising and show the potential of this approach, which should inspire to further investigation and work in the field.


