UPPSALA
UNIVERSITET

UPTEC F 17060
Examensarbete 30 hp
December 2017

Deep Convolutional Neural Networks for Real-Time Single Frame Monocular Depth Estimation

Jacob Schennings

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Deep Convolutional Neural Networks for Real-Time Single Frame Monocular Depth Estimation

Jacob Schennings

Vision based active safety systems have become more frequently occurring in modern vehicles to estimate depth of the objects ahead and for autonomous driving (AD) and advanced driver-assistance systems (ADAS). In this thesis a lightweight deep convolutional neural network performing real-time depth estimation on single monocular images is implemented and evaluated. Many of the vision based automatic brake systems in modern vehicles only detect pre-trained object types such as pedestrians and vehicles. These systems fail to detect general objects such as road debris and roadside obstacles. In stereo vision systems the problem is resolved by calculating a disparity image from the stereo image pair to extract depth information. The distance to an object can also be determined using radar and LiDAR systems. By using this depth information the system performs necessary actions to avoid collisions with objects that are determined to be too close. However, these systems are also more expensive than a regular mono camera system and are therefore not very common in the average consumer car. By implementing robust depth estimation in mono vision systems the benefits from active safety systems could be utilized by a larger segment of the vehicle fleet. This could drastically reduce human error related traffic accidents and possibly save many lives.

The network architecture evaluated in this thesis is more lightweight than other CNN architectures previously used for monocular depth estimation. The proposed architecture is therefore preferable to use on computationally lightweight systems. The network solves a supervised regression problem during the training procedure in order to produce a pixel-wise depth estimation map. The network was trained using a sparse ground truth image with spatially incoherent and discontinuous data and output a dense spatially coherent and continuous depth map prediction. The spatially incoherent ground truth posed a problem of discontinuity that was addressed by a masked loss function with regularization. The network was able to predict a dense depth estimation on the KITTI dataset with close to state-of-the-art performance.

Handledare: Gustav Jagbrant & Per Cronvall
Ämnesgranskare: Thomas Schön
Examinator: Tomas Nyberg
ISSN: 1401-5757, UPTEC F 17060

Populärvetenskaplig sammanfattning

Kamerabaserade säkerhetssystem för djupestimering och generell objektsdetektion är en viktig komponent i självkörande bilar (AD) och avancerade förarassistanssystem (ADAS). Denna avhandling utvärderar möjligheten att använda ett beräkningseffektivt neuronnät för att estimera djup i enstaka monobilder. För vanligt förekommande objekt på vägar så som bilar, fotgängare, cyklister och lastbilar finns det förtränade maskininlärnings-metoder som detekterar dessa objekt. Dessa metoder använder sig av stora mängder med annoterad data för att träna en klassificerare att hitta dessa specifika objekt i bilden. Ovanstående stämmer dock inte för generella objekt som kan representaras på en mängd olika sätt i olika former och kan förekomma i vägbilderna. I stereobaserade kamerasystem löses problemet med beräknandet av en disparitetsbild mellan höger och vänster kamera som räknar ut inversdjupet i bilden och kan på så sätt upptäcka generella objekt. För monobaserade kamerasystem har metoden Structure from Motion (struktur från rörelse) framgångsrikt används. Metoden fungerar genom att punkter i scenen spåras i en sekvens av bilder som sedan sammanfogas till en 3D representation av scenen. Denna metod kräver dock en sekvens av bilder där fordonet förflyttats genom sekvensen för att kunna konstruera 3D representationen av scenen. En annan metod som har ökat i populäritet under de senaste åren är att använda djupa faltningsneuronnät (deep convolutional neural networks) för att estimera djup i monobilder. Alla tidigare publicerade djupestimeringsmodeller med CNN:er har dock varit beräkningstunga.

Förhoppningen är att nätverket ska vara tillräckligt effektivt för att kunna köras på beräkningssvaga enheter och estimera djup i realtid för monobilder.

Systemen som tillsammans med kameror används i självkörande bilar idag är radar, LiDAR och sonar. Alla dessa system är dyra och kräver en större beräkningskapacitet för att behandla datan vilket också medför en prisökning. En implementation av djupestimering i monokameror skulle möjliggöra en prisreduktion vilket skulle kunna medföra en bredare spridning av automatiserade funktioner i bilar i fler olika prisklasser. Detta skulle kunna leda till att fler olyckor orsakade av mänskliga faktorn skulle kunna undvikas och förhoppningsvis rädda liv.

Acknowledgments

This thesis was proposed by Autoliv in Linköping, Sweden. I would like to extend my gratitude to my supervisors Gustav Jagbrant and Per Cronvall at Autoliv for their support and contribution to the project and the thesis report. I would also like to thank my subject reviewer Thomas Schön at Uppsala University for the assistance and encouragement provided during the course of this thesis.

On a personal note I would like to thank Karin Eklann for her invaluable never ending support, encouragement and care.

Contents

1. Introduction

1.1 Project description

1.1.1 Problem formulation

The goal of this thesis is to evaluate the possibility of generating a depth map estimate from a single monocular frame in real-time using a deep convolutional neural network (CNN). The work is carried out on behalf of Autoliv in Linköping, Sweden. The core properties of the network architecture is to be computationally lightweight and high performing to be able to output in real-time on a weak computational device. The network was initially designed for pixel-wise semantic segmentation with an encoder-decoder structure but is redesigned to estimate pixel-wise depth. The network architecture and loss function will be modified to solve the depth estimation problem. The network will be trained using supervised regression learning on the KITTI dataset [15] and parts of the internal Autoliv stereo dataset. The KITTI dataset contains stereo optical images together with LiDAR depth data used as ground truth. The Autoliv dataset contains stereo optical image data. The ground truth for the Autoliv dataset will be generated in advance by the stereo disparity algorithm used in Autoliv's commercial products. For both datasets the network will be trained using only images from one of the color cameras with simple data augmentation applied. The network will also be trained using the left and right frame separately as well as previous frames from the same camera as input. Low inference times are crucial so post processing methods presented in literature will be ignored to save computation overhead. No pre-trained network will be used and the network is trained in an end-to-end fashion to solve the regression problem.

1.1.2 Purpose

Object detection is a key component of Advanced Driver Assistance Systems (ADAS) and autonomous driving systems (AD). For objects common to road scenes such as pedestrians and vehicles, this is often based on supervised machine learning methods. By using large amount of annotated data classifiers are trained to detect specific object types within the scene. The above does, however, not hold for various object types such as debris that may show up in the scene. In stereo vision systems, this problem is resolved using stereo matching algorithms. However, for monocular vision systems, a notable approach is Structure from Motion (SfM). Image points are tracked over multiple 2D frames to construct a 3D structure of the scene and thereby extract depth. This method requires a sequence of images where the observer has moved between the frames, to derive the motion difference between images. One other approach that also has been used to determine scenic depth is Image based

modeling and rendering (IBMR). This method can reconstruct a 3D scenic representation of a scene using only a few 2D images taken at the scene from different points of view. The method combines the 2D images by tracking image points in the scene between images and derive the difference in perspective to reconstruct the 3D representation. Another approach which has gained more recognition in the literature in recent years is using deep neural networks for depth estimation [13, 12, 29, 14, 18, 27]. Previously, all such published work has been aimed at computationally expensive networks thus requiring expensive computational hardware. This thesis will however investigate the possibility of using a more efficient neural network architecture, suitable for embedded systems, to perform this task.

1.2 Background

Technology development has always been a central part of the vehicle industry and has improved the comfort, fuel consumption and especially, the safety in vehicles. The *passive* safety systems, such as collision zones, seat belts, airbags etc., are used to minimize personal damage and increase the probability of survival in the case of a traffic accident. The next step in the vehicle safety development is to prevent the traffic accident from ever happening by deploying *active* safety systems. By using more sophisticated monitoring and detection systems, such as several cameras, radar, LiDAR and infrared cameras, *active* safety systems can detect hazards on the road [2]. If a hazard is detected the system will alert the driver. In case of an inattentive driver or not enough time to react the system will perform actions such as automatic breaking or evasive steering to prevent the accident from occurring. A global deployment of *active* safety systems in future vehicles is of key importance for public traffic safety. A study made by the U.S. Department of Transportation in 2015 showed that on estimate the main cause for traffic accidents involving light vehicles was human error with 94% ($\pm$2.2%). Only 2% ($\pm$0.7%) of the accidents were caused by vehicle malfunction and 2% ($\pm$1.3%) by environmental effects such as slippery roads. The last 2% were due to unknown critical reason [42].

Depth Estimation Before CNNs

Many different approaches have been utilized before deep learning based models to predict scenic depth. Methods such as Structure from Motion (SfM) and Optical Flow determine depth from a sequence of images. These methods track groups of pixels in the image and use the difference between consecutive frames to construct a 3D structure from 2D images. These methods, however, demand a sequence of images and cannot estimate depth from a single image. The depth estimation method Make3D, introduced by Saxena *et al.* [40], is an image patch-based model that estimates depth through 3D estimation of segmented image patches. It was one of the first supervised learning-based monocular depth estimation methods. The depth prediction was modeled in a Markov Random Field and they used multi-scale handcrafted texture features to determine depth in the local patches. The Make3D model greatest struggles are predicting depth for thin objects and and also the global scale of the examined scene. This occurs since only local predictions are made for the image patches and there is no global context available to combine the patches.

The common problem for most of the previous methods is that they rely on hand-crafted features which are time consuming for engineers to design and evaluate. To move away from hand crafted features machine learning was used to find feature representations within images since the method acts as a representation learner. Since a computer can process the same image thousands of times it can try thousands of different features which streamlines the process of finding robust feature representations. In doing so, the computer can evaluate and hopefully find low-level feature representations that were unthought-of by feature construction engineers.

Brief Deep Learning History

Over the past decade the utilization of machine learning methods, in particular deep learning, has increased tremendously. Reasons for this could be that the methods are computationally heavy and require large datasets, two requisites that now are available. Deep convolutional neural networks have proven to be a good approach to a multitude of challenging computer vision problems. In 1998 Yann LeCun et. al [31] successfully implemented a CNN architecture, LeNet-5, to classify handwritten characters on bank cheques that outperformed the previous state-of-the-art methods. This network was trained and evaluated on 32×32 single channel input images with one character per image. The network contained $\sim 60,000$ learnable parameters and took 2-3 days to train on a server using a single 200 MHz CPU. Larger network architectures were not considered since the training time for these was not feasible at the time. However, the uprise in the development of graphic processing units (GPU's) has opened up the possibility of performing massive parallel computations in real-time. These are perfectly suited for training and developing CNN's. The method resurfaced in 2012 when the CNN AlexNet [26] won the image classification challenge ILSVRC-2012 [39] by outperforming the second best competitor with 10.9%. The network contained 60 million learnable parameters and was, thanks to the modern GPU, possible to train on two NVIDIA GTX 580 GPU 3GB during a period of 5 to 6 days. With the increased computational performance and huge datasets available the development and research of CNN's took off.

1.3 Prior Work

Deep convolutional neural networks have proven to be very capable in solving the problem of depth estimation from a single monocular image. These deep learning based methods have solved the depth estimation problem using either; a supervised learning scheme [13, 12, 29], an unsupervised learning scheme [14, 18, 49] or a semi-supervised learning scheme [27]. The learning is unsupervised in terms of it not requiring a depth ground truth. Since the task of supervised depth regression is closely related to the semantic segmentation task [29] many of the depth estimation networks are based on high performing semantic segmentation networks.

Eigen *et al.* [13] were the first to solely use a CNN approach (based on AlexNet [26]) to solve the depth regression problem and output a dense depth map prediction. They used a two stage architecture, the first stage produced a coarse output for global perception and the second stage produced a fine output for local detail refinements. This model successfully outperformed the state-of-the-art depth estimation

method by Ladicky *et al.* [28]. However, the network proposed by Eigen *et al.* [13] was computationally heavy and consisted of a multi-scale processing architecture.

Eigen *et al.* [12] showed that a network architecture designed for semantic segmentation is very capable in estimating scenic depth. They used these properties in their model to predict depth, surface normals and semantic segmentation using only different final layers. For the depth estimation task the network predicts pixel-wise real valued quantities to produce the dense depth map.

In order to refine the predicted depth map Laina *et al.* [29] used an architecture (based on ResNet-50 [20]) with fully convolutional layers to upsample the image. The fully convolutional layers use up-sampling residual modules instead of fully connected layers which proved to be a better way for feature learning. Using fully convolutional layers instead of traditional fully connected layers proved more suitable for tackling high dimensional regression problems such as depth estimation. The model was trained end-to-end and did not use any post-processing techniques. The depth estimation problem was posed as an unsupervised problem [14, 18] where the ground truth was not used during the training process.

Garg *et al.* [14] pose the depth estimation problem as an image reconstruction problem using an autoencoder based layout based on the AlexNet architecture [26]. The method only requires image pairs from a stereo camera and use the photometric, or reprojection, error from one input image warped into the other as the loss. They use the image reconstruction and known base line of the stereo camera to calculate the scenic depth. However, the method struggles with a non-fully differentiable image formation model. This is compensated by implementing a Taylor approximation to linearize the loss. This hampers the performance of the network since the approximation is more demanding to optimize.

Godard *et al.* [18] utilized an unsupervised approach, similar to Garg *et al.* [14] but with a left right consistency. They show that accounting only for the image reconstruction problem loss results in poor quality predictions. This is solved by introducing left-right consistency that enforces the disparity image to be consistently produced relative to the left and right image. This method results in a more robust and precise depth map prediction.

The depth estimation problem was posed as a semi-supervised problem by Kuznietsov *et al.* [27] where they use an unsupervised and a supervised learning scheme to train their model. The method is based on a ResNet-50 encoder-decoder CNN architecture. They use an unsupervised direct image alignment loss to enforce the network to produce a photoconsistent dense depth map along with a sparse ground truth for supervised learning. This method is combined with a supervised depth regression to refine the results. The method produce detailed state-of-the-art depth estimation maps of the KITTI dataset [15].

Several other methods using different constellations of Conditional Random Field (CRF) with superpixels have been utilized in solving the depth estimation problem [46, 33, 32, 34]. These approaches are however not of interest since they utilize image pre-processing which is not ideal for real-time implementations. The problem has also been posed as a classification task [5, 28] where pixel-wise uncertainty measurements are available for the networks predictions. These methods discretize the depth interval into bins and each pixel is classified to one of these bins which could result in a crude depth interval.

All of the previously used depth estimation methods without image pre-processing

utilize large network architectures with many millions of parameters. This is not suitable for implementation on an embedded system or a low power device since the lack of storage and computational power will render these methods useless.

2. Theory

2.1 Monocular depth estimation

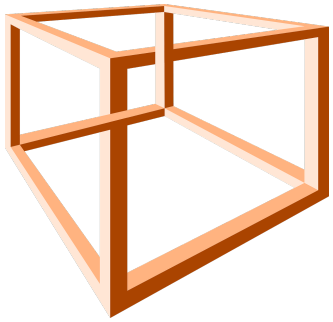
Extracting depth information from a scene is a trivial task for humans. This is something that is self-evident for most humans since this is performed by the visual cortex and is done unconsciously. The visual cortex utilizes stereopsis and monocular cues to estimate depth within a scene [22]. However, this depth information is only perceived and not absolutely determined. This means that a human is very skilled in reading relative scenic depth information and determine how an object is related to its surroundings. The problem gets increasingly more difficult for a human if the absolute depth is to be determined, especially if only monocular cues are present. To understand the complexity of monocular depth estimation some of the monocular cues utilized by the visual cortex needs to be reviewed.

2.1.1 Human vision

When observing a scene the visual cortex uses stereopsis and monocular cues to determine scenic relative depth. When stereopsis is unavailable, e.g. when observing an image or having a monocular vision impairment, the visual cortex uses different monocular cues to interpret depth from scenes [22]. The general perspective of a scene could be used to explain why small objects are perceived as far away. The linear perspective of a scene helps the vision system to determine depth magnitude by analyzing the convergence of parallel lines, e.g. a long straight road or a tall skyscraper (Figure 2.1a). The perceived distance of familiar objects, e.g. a car, can be determined by using the object size in conjunction with the visual angle covered by the object on the retina. The vision system also uses relative size of similar or familiar objects in a scene to determine the objects perceived relative distance to one another. The visual system can also determine depth by examining the texture gradient within a scene, for example when observing a gravel road. The road surface close to the observer is rich in texture details and the further away the observer looks the level of detail decline. This implies that the observer can use the texture gradient to determine perceived distance in a scene. These and other monocular cues make depth perception a routine feature for humans. However, these monocular cues can also be manipulated to disorder the visual cortex into incorrectly determine the perceived depth of a scene. Examples of these are optical illusions where for example the perspective of familiar objects are displaced, presented in Figure 2.1b.

Even though humans are very skilled at perceiving depth the problem of absolute depth estimation, is still a challenging task for most humans. The problem complexity increases even more when the depth estimation problem is presented to a computer.

(a) Linear perspective example



(b) Optical perspective illusion

Figure 2.1: Examples of when monocular cues are useful (2.1a) and when they are manipulated (2.1b). Images under CC0 license [9].

2.1.2 Computer vision

Computer vision is a very large field of research that cover fields such as biology, physics, psychology, mathematics, engineering and computer science. In order to solve computer vision problems it is important to understand how the vision system operates in animals and how objects are recognized by them. This knowledge is then to be implemented in a computer with hardware and software to solve the vision tasks. This area of research is very youthful compared to other research areas and has in recent years catched up with human performance in some specific tasks [21]. Computers have been utilized to solve image related problems for many years with a variety of methods. One of the first attempts of tackling the computer vision problem was in 1966 during the "The summer vision project" at MIT [37]. The project goal was to construct a system that could divide parts in an image scene into different classes such as objects, background or chaos. The project coordinator thought it was possible, using a group of summer workers, to learn a computer to describe "what it saw" during the course of a summer. This problem formulation proved to be a bit optimistic.

2.1.3 Digital image representation

To present a monocular scene to the computer it is captured by a visual recording media and stored in an image file. The image constrains the scene to a spatially fixed and finite representation of the intensities in the scene. The intensities are discretized to numeric values, pixels, and stored in a two dimensional spatially ordered array $X \times Y$. Allowing more information from the scene to be stored, such as color, each pixel location has several channels D associated to it that create a 3D volume with dimensions $X \times Y \times D$. A color image stores the light intensities from the scene for different wavelengths in each channel. For a regular color image these wavelengths correspond to the red, green and blue colors (RGB). Combining these RGB channels renders a color image.[1] The bits per channel determine the total amount of color

[1]The actual combination of the color channels is non trivial and is implemented by using the Bayer pattern on the imaging sensor and a demosaicing algorithm to create the color blend in each pixel in the multi-channel image.

variations that can be stored in each pixel. E.g. a 24-bit RGB image, with 8-bits per channel, can represent $(2^8)^3 = 16,777,216$ different color variations in each pixel. The image can thus represent the scene, containing the monocular cues, and store enough visual information for humans to estimate the depth within the scene. This implies that the same should also be possible for a computer.

2.2 Machine learning

There exists a variety of machine learning methods that can be used for solving many different types of problems. These methods have been used to solve a variety of hard problems that previously lacked any robust solutions. The success of these methods is primarily the methods ability to optimize the internal parameters through training and evaluation schemes in order to increase performance. There are two main schemes in machine learning; supervised and unsupervised learning. The supervised learning is based on the usage of a known output, or ground truth, during the training process. The unsupervised learning is based on using only input data to train the network without the utilization of a ground truth in the training process. The focus will lay on the supervised learning scheme that will be used to solve the depth regression problem.

2.2.1 Supervised learning

A supervised learning method is based on optimizing a model with known input and output data to generate predictions for input data without a known output. The model gets an input x and makes a prediction $\hat{y}$ that is evaluated toward the known output or ground truth y in the training process. The goal of the training is for the model to learn a mapping function of the input data to the output data.

2.3 Neural networks

Artificial neural network are inspired by the biological neuron structure that constitute the brain of animals. These systems learn to solve a problem, without task-specific programming, by reviewing many examples. In an image classifier the network learns the characteristics of each class in the set to classify new input images to their corresponding classes. To train and validate the performance, the dataset used is split into training, validation and test set. A neural networks consists of an input layer, an output layer and a chain of connected layers in-between called hidden layers. These layers are called hidden since they are not presented to the user. Neural networks that contain more than a few hidden layer are called deep neural networks.

Within the network there are two different type of layers; learnable and non-learnable layers. The learnable layers contain learnable parameters that represent the majority of parameters within the network architecture. The total number of parameters and the parameter distribution over the layers can vary a lot depending on the network architecture. The network iteratively updates these parameters during the network training via backpropagation to increase the prediction performance. Each iteration is also called a weight update since the network weights are

updated for every iteration. The total number of weight updates are determined by the training set size and the number of times the network examines the whole training set, the number of epochs. The non learnable layers only performs different data treatments without extracting information and can thus not be optimized, or learned.

Prediction model

A basic neural network with a supervised learning scheme predict an output $\hat{y}$ for an unknown input x^* by training the network with known input x and output y data. The linear regression problem is defined by

$$\hat{y} = Wx + b \tag{2.1}$$

where x is the input, W the weight, b the bias term and $\hat{y}$ the predicted output. The network is then trained in order to output a prediction $\hat{y}$ of y of preferably high accuracy. The training principle is the same for image related problems. The network still tries to optimize the weights and bias terms to output an as good as possible real valued prediction. The algorithm used in neural networks is defined in Algorithm 1.

Algorithm 1 Neural network training algorithm

1: **Input:** Input x, Loss function $\mathcal{L}$, Ground truth y, Epochs K, Weights W,
2: **Output:** Network prediction $\hat{y}$
3: **Dataset** Ω**:** Training set $\Omega_{train} \subset \Omega$, Validation set $\Omega_{val} \subset \Omega$, Test set $\Omega_{test} \subset \Omega$ with $\Omega_{test} \cap \Omega_{val} \cap \Omega_{test} = \emptyset$
4: **Initialization:** Random weights, epochs count $i = 0$
5: **while** $i \leq K$ **do**
6: Forward propagate $x \in \Omega_{train}$ through network
7: Generate prediction y at output
8: Evaluate prediction $\mathcal{L}(y, \hat{y})$
9: Backpropagation of loss gradient $\frac{\partial \mathcal{L}(y,\hat{y})}{\partial W}$
10: Update W and b with gradient based optimization method
11: Evaluate network on $x \in \Omega_{val}$
12: **end while**
13: Generate predictions of test set Ω_{test} with the network with best performance on validation set
14: Evaluate performance of network predictions

2.3.1 Loss function

The loss function is used to evaluate the performance of the network prediction during training. The network prediction $\hat{y}$ is evaluated towards the ground truth y in the loss function where the prediction error is calculated. Different tasks require different loss functions. For image related dense prediction tasks the loss is evaluated pixel-wise. The loss is only evaluated on the pixels in the prediction with corresponding pixels in the ground truth. The rest of the pixels in the prediction are ignored and the pixel areas are treated as “do not care”. This masked loss function

is used to train the CNN to output a dense spatially coherent prediction using a sparse spatially incoherent ground truth.

For depth estimation on single monocular images there are a variety of different loss functions presented in the literature. These loss functions evaluate the prediction error on different scales and are suitable for different types of ground truth. The proposed network solves a regression problem with a linear ground truth. There were three loss functions of interest; the $\mathcal{L}_1$ norm, the $\mathcal{L}_2^2$ norm and the berHu norm that is a combination of $\mathcal{L}_1$ and $\mathcal{L}_2^2$ norms. The difference between the two norms is presented in Figure 2.2.

The berHu norm is a reversed Huber norm and was used by Laina *et al.* [29] and Kuznietsov *et al.* [27]. Eigen *et al.* [13] used a scaled version of the $\mathcal{L}_2^2$ norm of the logarithm of the prediction and ground truth in order to use a scale-invariant loss function. Eigen & Fergus [12] used use a combination of the pixel difference squared and the image difference squared in their loss function. They also use the horizontal and vertical image gradients as a regularizing factor in the loss function, penalizing high image derivatives.

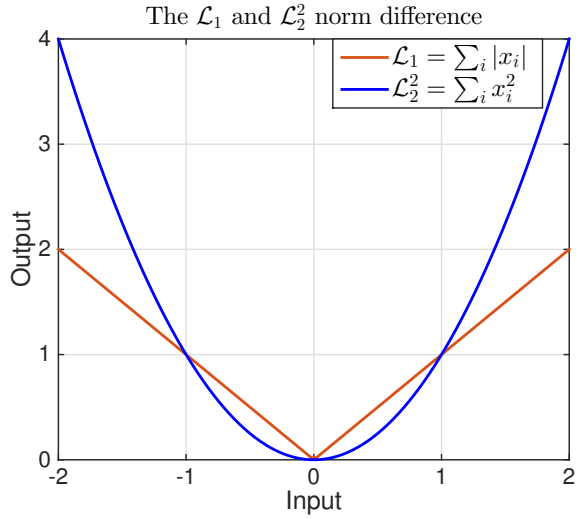


Figure 2.2: The $\mathcal{L}_1$ and $\mathcal{L}_2^2$ norm of the prediction error x.

$\mathcal{L}_1$ norm

The $\mathcal{L}_1$ norm or mean absolute error is defined as

$$\mathcal{L}(\rho(\mathbf{x}), Z(\mathbf{x})) = \frac{1}{n}\sum_{i=1}^{n} |\rho(x_i) - Z(x_i)| = ||\rho(\mathbf{x}) - Z(\mathbf{x})||_1 \tag{2.2}$$

where n are the pixels with ground truth in the prediction, $\rho(\mathbf{x})$ is the network prediction and $Z(\mathbf{x})$ is the ground truth. The $\mathcal{L}_1$ norm uses linear regularization to minimize the loss of the prediction. It does not produce a high loss for large prediction errors since the norm is linear, as seen in Figure 2.2.

$\mathcal{L}_2^2$ norm

The squared $\mathcal{L}_2$ norm or mean squared error is defined as

$$\mathcal{L}(\rho(\mathbf{x}), Z(\mathbf{x})) = \frac{1}{n}\sum_{i=1}^{n}(\rho(\mathbf{x}_i) - Z(\mathbf{x}_i))^2 \tag{2.3}$$

where n are the pixels with ground truth in the prediction, $\rho(\mathbf{x})$ is the network prediction and $Z(\mathbf{x})$ is the ground truth. The norm is squared to simplify the differentiation of the loss function in the backpropagation. The $\mathcal{L}_2$ norm uses a squared regularization to minimize the loss of the prediction. The loss has a stronger penalty than the $\mathcal{L}_1$ norm for errors > 1 and weaker penalty for errors < 1. The norm can cause problems for outliers and measurement noise since these can produce large errors. The problem of outliers and noise is however reduced by the mask function. Only the pixels from the ground truth are evaluated thereby removing the risk of outliers in the loss evaluation. This does however not reduce the risk of erroneous measurements from the ground truth itself.

berHu norm

The berHu norm or the inverse Huber norm introduced by [29] is a combination of the $\mathcal{L}_1$ and a scaled $\mathcal{L}_2^2$ norm. The berHu norm $\mathcal{B}$ is defined as

$$\mathcal{B}(d) = \begin{cases} |d| & |d| \leq c, \\ \frac{d^2+c^2}{2c} & |d| > c. \end{cases} \tag{2.4}$$

where $d = \rho(\mathbf{x}) - Z(\mathbf{x})$. The parameter $c = \frac{1}{5}\max_i(|d|)$, where i indexes all the pixels with information in the ground truth image, corresponds to 20% of the highest prediction error in the batch. The scaling value of the highest prediction error was proposed by Laina *et al.* [29]. The berHu norm tries to implement both the linear and non linear properties of the $\mathcal{L}_1$ and $\mathcal{L}_2^2$ norms. The threshold parameter c both scales and choose the inflection point for which loss function to be used in order to utilize the positive effects of both norms.

2.3.2 Gradient descent

The gradient descent method is a gradient based iterative optimization method that is used to find the minimum of an objective function $F(x)$. This is achieved by iterative incremental stepping of the point x in the negative gradient direction $-\nabla F(x)$ to move toward the function minimum. The gradient descent is defined by

$$x_{i+1} = x_i - \gamma \nabla F(x_i) \tag{2.5}$$

where x_i is the point where the objective function is currently evaluated, $\nabla F(x_i)$ is the gradient of the objective function and γ is the step length parameter. This method minimizes the function by iteratively stepping closer to the minimum. For neural networks the method is used to minimize the loss function with respect to the network weights in order to increase the network performance.

Stochastic gradient descent

The stochastic gradient descent (SGD) method is an approximation of the gradient descent method for a sum of differentiable functions $F(x) = \sum_i F_i(x)$. The stochastic gradient descent is defined by

$$x_{i+1} = x_i - \frac{\gamma}{n} \sum_{j=1}^{n} \nabla F_j(x_i). \tag{2.6}$$

In order to calculate the gradient of the objective function all of the summand gradients are computed and summed. If the summands are simple functions and n is reasonably small, the summation does not pose a computational demanding problem. However, if the summand functions are more complex and if n is larger, each step in the gradient descent is much more computationally complex requiring all n summands being stored in memory. The SGD circumvents this issue by stochastically sampling a batch of the summand functions and calculate their gradients in order to approximate the derivative of the objective function. By stochastically selecting a smaller batch of summands the computational cost is greatly reduced and the approximation results in an accurate solution. The SGD is a key factor in realizing neural network optimization with vast amount of training data that otherwise would pose a very computationally complex problems.

2.3.3 Backpropagation

Backpropagation is one of the key methods in neural network development since it simplifies the computation of local neuron gradients that are used in the network optimization. Backpropagation use the chain rule to decompose the gradient of the loss w.r.t. the weights at the network output into local neuron gradients. These local gradients are used in the optimization of the network weights. First the network predicts an output for an input by forward propagating the input through network. The prediction is then evaluated towards the ground truth value at the network output with the given loss function. The gradient of the loss function w.r.t. the network weights is also computed at the output and is then sent back through the network via backpropagation. During the backpropagation the gradient it is decomposed into local neuron gradient contributions. Each local neuron gradient is then used in the gradient descent optimization to minimize the error contribution from each neuron. By applying gradient descent optimization together with backpropagation the network prediction accuracy increases.

2.4 Convolutional neural network

Deep learning based models such as deep convolutional neural networks, or CNN's, have proven to be a good approach when working with image related problems. The CNN architecture is very efficient in interpreting and extracting image information for tasks such as classification [39, 15] or semantic segmentation [8, 41] when compared to other non-CNN architectures. This property in combination with the learning capability provides a great base for image based learning. The CNN solves vision based problems by finding low level feature representations for different objects in the image. These features are learned by the CNN and not hand-crafted

which is the major difference between deep learning methods and previous computer vision methods. The features are iteratively learned by the network during the training process. These features are found through an architecture containing connected weights that extract information from the image through primarily convolution operations. A CNN usually contain millions of weights that are used to represent individual features within the network. Krizhevsky *et al.* [26] showed in the ImageNet image classification competition [39] in 2012 that the proposed AlexNet CNN was superior in solving the image classification problem. This has also proven to be the case for other computer vision tasks such as depth estimation [13, 12, 29, 14, 18, 27] and semantic segmentation [3, 6, 35] that both generate pixel-wise predictions. Despite the mentioned architectures high performance problems still remain. The majority of the architectures are very computational heavy which means that they require powerful hardware to operate preventing implementation on computationally weaker systems.

2.5 Network components

The overall design of CNN's can vary a lot, but almost all of them share the same set of components distributed over the network layers. Each CNN is trained using one or more graphics cards (GPU's) since the parallelization capability of the GPU perfectly fit the network training. The training is carried out by feeding mini-batches of images, the stochastically selected summands in the SGD, to the network. These batches reduce training time due to parallelization and also aid the network convergence by implementing batch normalization, covered in section 2.5.5. The size of the mini-batch is determined by the user and is limited by the memory on the GPU.

The network components presented below are all implemented in the network architecture used in the thesis.

2.5.1 Convolutional layers

Different kind of convolutional layers are presented in this section. All of the layers employ the same convolution principle as the regular convolutional layer but with different filter spatial representations.

Regular convolutional layer

The convolutional layer is the primary learnable layer for the CNN architecture. Each layer is constructed by a number of filters, or kernels, that contain the weights W and a bias term b. The filters extract feature information from the input data, either the input image in the first layer or an output from a prior layer. Each filter is defined by a spatial size $F \times F$ and a stride S and produce a feature map when convolved with the input. The spatial size of the filter determines the number of weights in and the receptive field of the filter. The stride is the step length that the filter uses when it convolves the image. The produced feature map dimensions are determined by the filter stride, spatial size and padding. For an input image $X \times Y$, the feature map dimensions are $X_{conv} = (X - F)/S + 1$ and $Y_{conv} = (Y - F)/S + 1$. The regular convolution with reduced spatial dimension is presented in Figure 2.4.

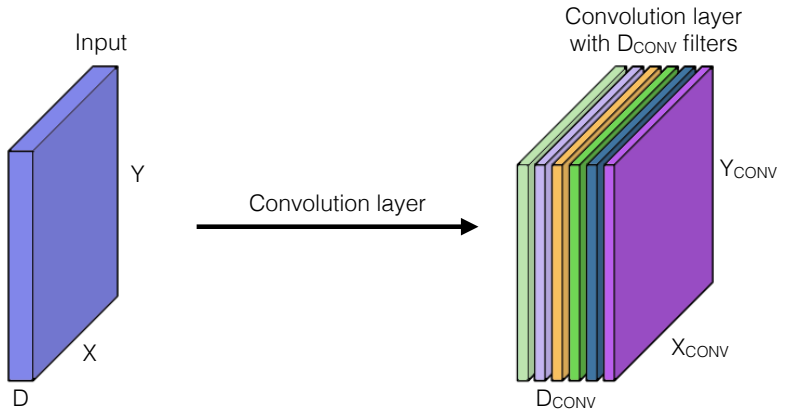


Figure 2.3: Resulting volume of feature maps for input after convolution with D_{conv} filters.

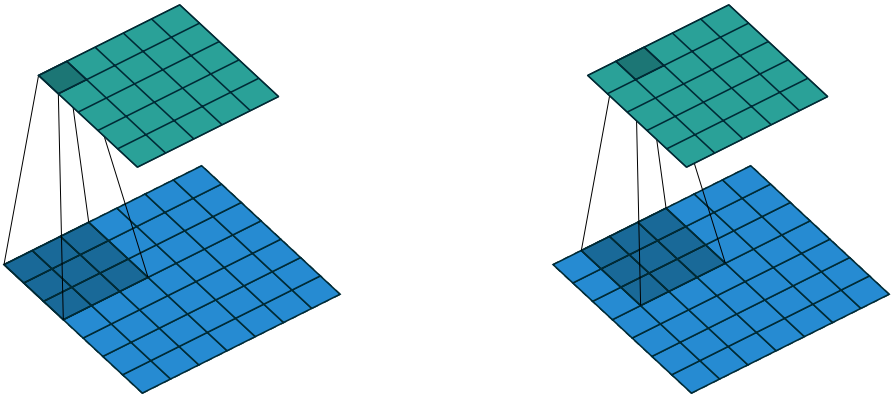


Figure 2.4: The regular convolution of the input (lower) with filter size 3×3 and stride 1 producing the feature response map (upper) with lower spatial resolution. Figure provided by [11].

The feature map is then sent through an activation function (reviewed in Section 2.5.2) that transforms it to an activation map with the same spatial dimensions. These are stored in a volumetric shape with D_{conv} channels determined by the number of filters in the layer. The number of channels in each layer D_{conv} also corresponds to the number of bias terms b in the layer. This volume of activation maps is the output from the convolutional layer as seen in Figure 2.3. The number of parameters P in the convolution layer with input $X \times Y \times D$ and D_{conv} filters with spatial size $F \times F$ and bias term b is determined by

$$P = (F \times F \times D) \times D_{conv} + D_{conv}. \tag{2.7}$$

For Figure 2.3 this would result in $(5 \times 5 \times 3) \times 6 + 6 = 450$ learnable parameters. It can be observed in Figure 2.4 that the activation map is slightly spatially smaller than the input. This effect can be avoided, if wanted, by using zero padding on the input. The input borders are padded with zeros that increase the spatial dimensions with P on each side. The spatial dimension of the output for a zero padded input

is determined by $X_{conv} = (X - F + 2P)/S + 1$ and $Y_{conv} = (Y - F + 2P)/S + 1$. With proper zero padding the output will have the same spatial dimensions as the input, presented in Figure 2.5.

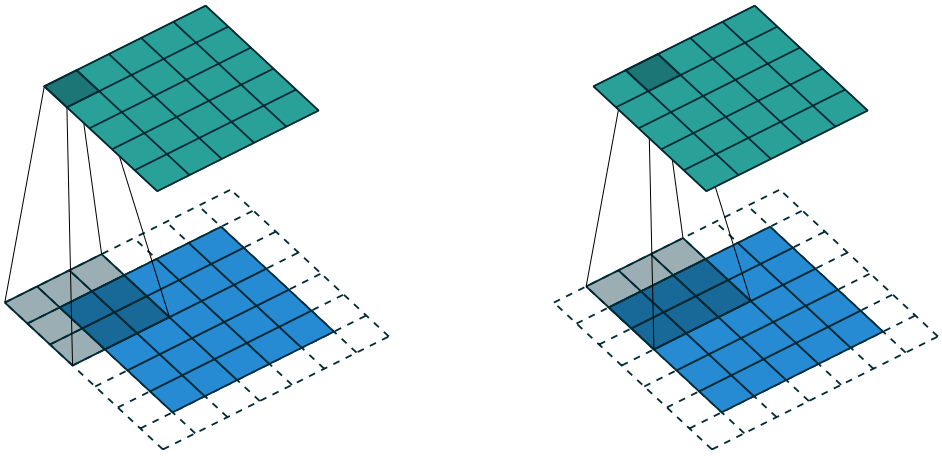


Figure 2.5: The regular convolution of the zero padded input (lower) with filter size 3×3 and stride 1 producing the feature response map (upper) with equal spatial dimensions. Figure provided by [11].

Dilated convolutional layer

The dilated, or à trous, convolutional layer was introduced roughly simultaneously by [47, 6] and is a method used to increase the receptive field of the network without increasing the number of weights. To get a larger receptive field in a conventional CNN architecture pooling layers are used to reduce spatial information redundancy. First the image is downsampled in the pooling layer and then convolved within the convolutional layer. Then, if the network performs a dense prediction task, the feature response pixels of the output are mapped to the spatial dimensions of the input image. This could be performed by a reversed pooling operation or a feature dilation with zeros. However, the number of feature responses now only account for a fraction of the input image positions resulting in a sparse or interpolated feature response map as seen in Figure 2.7. The sparse representation could in turn be interpolated to a dense representation. However that would alter pixel values and further add computations. The convolution kernel could also be spatially larger to increase the receptive field but this would increase the number of weights and increase the computational complexity of the problem.

The dilated convolutional layer addresses the receptive field issue in another manner. The layer is based on the "algorithme à trous" concept where the downsampling is removed and the filter is upsampled. The upsampling is executed by inserting zeros, or "holes", in the filter which increases the spatial dimensions. The dilated filter then covers a larger perceptive field of the input image without adding more weights. Thus achieving the same enlarged receptive field as a spatial downsampling of the image would give while simultaneously preserving information in the image. The spatial dimension of the dilated filter is controlled by the *rate* parameter that determines the stride between each weight thus the amount of zeros dilating

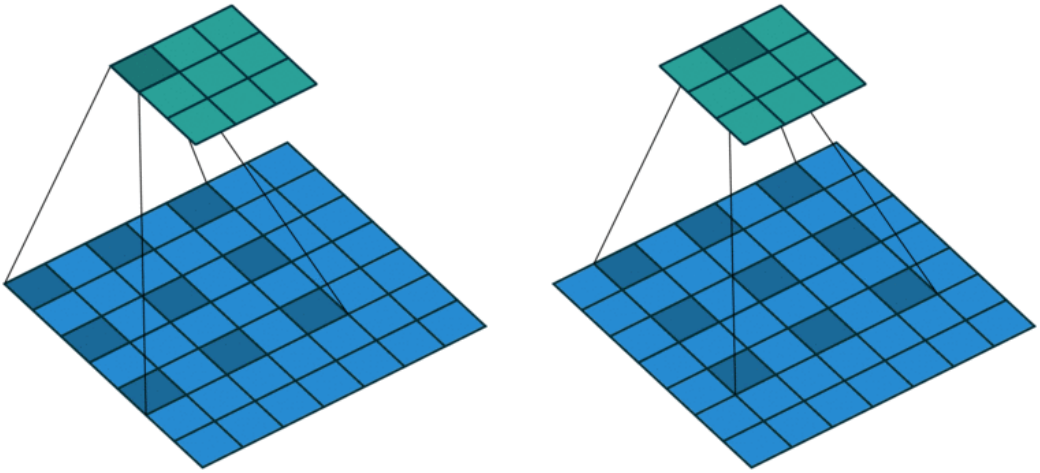


Figure 2.6: The dilated convolution kernel with kernel size 3×3 and $rate = 2$ utilizing a receptive field of 5×5. The number of parameters are constant in the filter but the filters perceptive field increases with the rate parameter.

the filter. If the filter is 1D this means that $rate = 1$ gives the regular convolution since the step length would be 1 and there would be no space between the weight. The $rate = 2$ results in a step length of 2 resulting in one zero between the filter weights. The $rate = 2$ is used in Figure 2.6 where one zero is inserted between each filter weight. The implementation of dilated convolution layers allows for explicit control of the resolution at which feature responses are computed without adding weights.

Asymmetric convolutional layer

The asymmetric convolution layer introduced by Szegedy *et al.* [43] is built upon the idea that regular convolution kernels contain redundancies. These are diminished by using factorized convolutions, or asymmetric convolutions. For a regular symmetric convolution, of say 3×3, the filter is factorized into a series of two asymmetric filters 3×1 and 1×3. This efficiently reduces the total number of parameters for the convolution from $3 \times 3 = 9$ to $3 \times 1 + 1 \times 3 = 6$ weights while maintaining the receptive field of the regular symmetric filter. The module also adds a non linearity between the filters which increases the richness of the features learned by the asymmetric convolution module.

Deconvolution layer

The deconvolution, or backwards convolution, layer is used to upsample the feature map in-network without interpolation. The deconvolution layer was implemented by Long and Shelhamer *et al.* [35] to perform in-network upsampling, presented in Figure 2.8. They found it to be a fast and efficient method for learning dense predictions. The layer is simply a reversed convolution layer where the feature map is dilated with zeros and then convolved. This produces a spatially larger output for the deconvolutional layer without using regular interpolation. These layers stacked

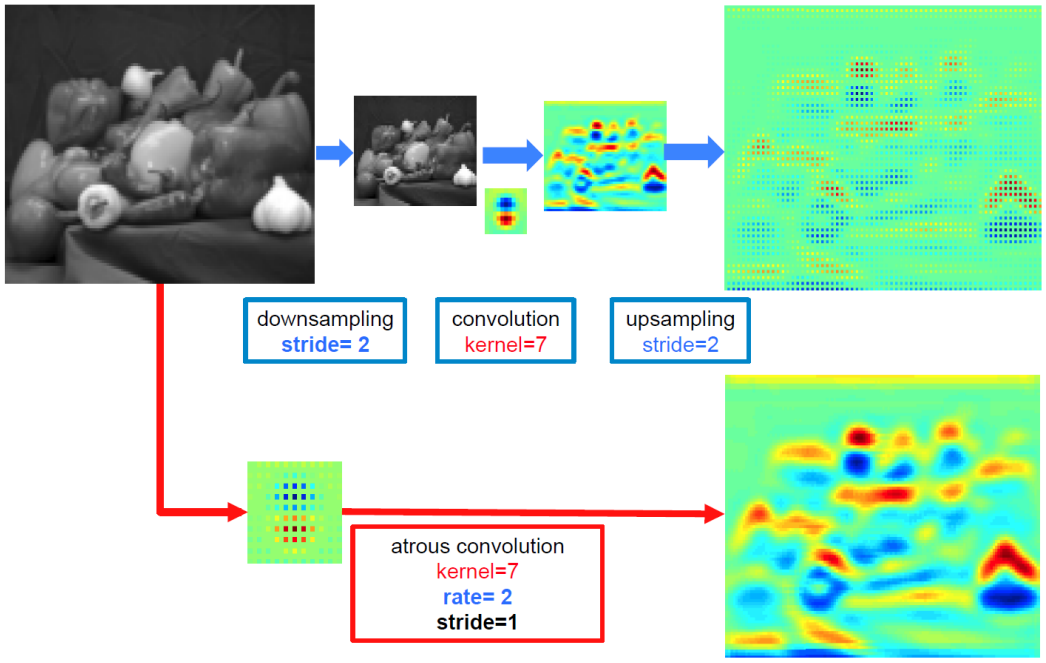


Figure 2.7: The sparse feature response map produced by a regular CNN architecture with convolution and pooling layers and the dense feature response map produced by the atrous convolutional layer (Reproduced from [6]).

together with activation functions within the network decoder can even learn non-linear upsampling. By using deconvolution layers the network is able to generate higher detailed and spatially larger output predictions instead of interpolating the output to be spatially larger.

2.5.2 Activation functions

Activation functions are used in the network to transform the neuron input response and add a non linearity to the otherwise linear network architecture. The activation function in conjunction with the neuron input response control the neuron output, or the neuron activation.

There have been several activation functions used in the past. During recent years the Rectified linear unit (ReLU) and different variations of it have become the first choice for neural networks [30]. The ReLU was introduced by Hahnloser *et al.* in 2000 [19] and first used in deep neural networks by Glorot *et al.* [17] in 2011. The primary reason for the popularity is the increased convergence rate and simpler gradient. This results in reduce training time with similar accuracy of the previously popular sigmoid activation function.

Sigmoid

In the beginning of the deep neural network research area a commonly used activation function was the sigmoid function

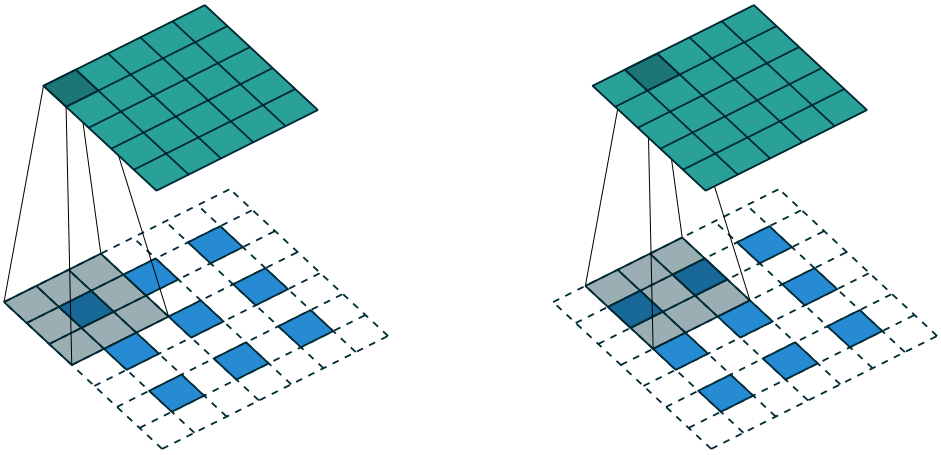


Figure 2.8: The deconvolution layer applied to the input with spatial resolution 3×3 (lower) produce an output with spatial resolution 5×5 (upper) with filter spatial size 3×3, stride 1 and rate 1. Figure provided by [11].

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{2.8}$$

It was primarily used since it has a simple derivative but also that it has similarities with biological neuron activations in mammal brains. The Sigmoid function takes the input value x and squeezes it onto the interval $[0, 1]$. The function value then represents the non-linear behavior of a biological neurons firing rate. For no activity at all it outputs 0 and for the maximum possible firing rate it outputs 1. The sigmoid function has a non-negative bell shaped first derivative $f'(x) \to 0$ for $x \to \pm\infty$ as observed in Figure 2.9. The function saturate and kill gradients at $f(x) = 0$ or 1 which causes a major issue in the neural network. Neurons with saturated gradients will not have their weights updated in the network training. The backpropagation of the loss will not be allowed to flow through neurons with saturated gradients causing their weights not to update. Another problem of the sigmoid function is that it outputs non-zero centered data. This causes a problem during the backpropagation since consecutive layers would be receiving non-zero centered data which affects the gradient descent optimization. This would result in either all positive or negative gradients of the weights causing a non wanted oscillating dynamics in the backpropagation and weight update procedure.

The tanh function is an alternative to the Sigmoid function. It outputs zero centered data thus removing the unwanted oscillation during the weight update but it still suffers from possible saturated gradients.

Rectified Linear Unit

The rectified linear unit (ReLU) is a computationally simple activation function that is for an input x defined as

$$f(x) = \max(0, x). \tag{2.9}$$

The function performs a threshold operation that only activates, or throughputs,

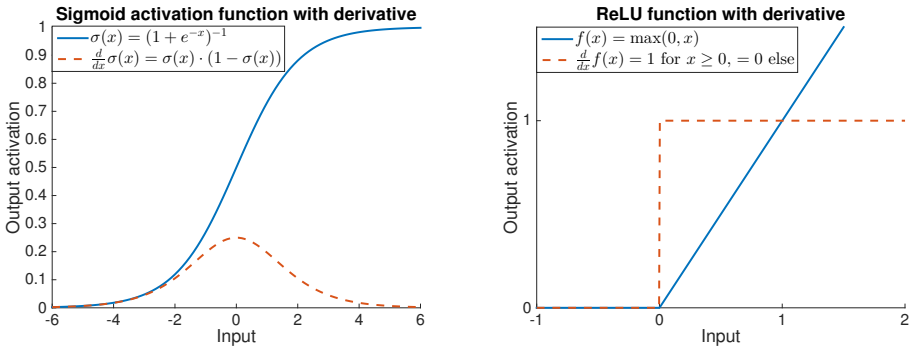


Figure 2.9: The sigmoid function and the ReLU function complete with first order derivatives.

for positive values. As of 2015, the ReLU activation function was one of the most commonly used activation functions [30]. Reasons for this are primarily the differentiability, the reduced risk of saturating gradients and a significantly improved convergence rate. The difference between the ReLU and the sigmoid function values and derivatives are shown in Figure 2.9.

The ReLU's reduced complexity results in a more simple gradient that in turn reduce the computational cost for training the network, without negatively affecting the network performance [26]. The improved convergence rate for image classifying CNN's using ReLU's was shown by Krizhevsky *et al.* [26] in the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) in 2012. By implementing ReLU activation functions in the network architecture the convergence rate increased with 6x when compared to the previously used sigmoid/tanh functions.

The ReLU, in contrary to the other saturating nonlinearities, has a constant gradient for $x \geq 0$ that eliminates the problem of gradients vanishing as x increases. However, if the ReLU is exposed to a negative input it will not activate resulting in a zero output and a zero gradient during backpropagation. The neuron could then be considered as 'dead' since it no longer would be updated during the training process and not be a part of the network. This problem was taken care of by allowing neurons to have a small output 'leakage' for negative input values. Resulting in a preserved small local neuron gradient in the backpropagation.

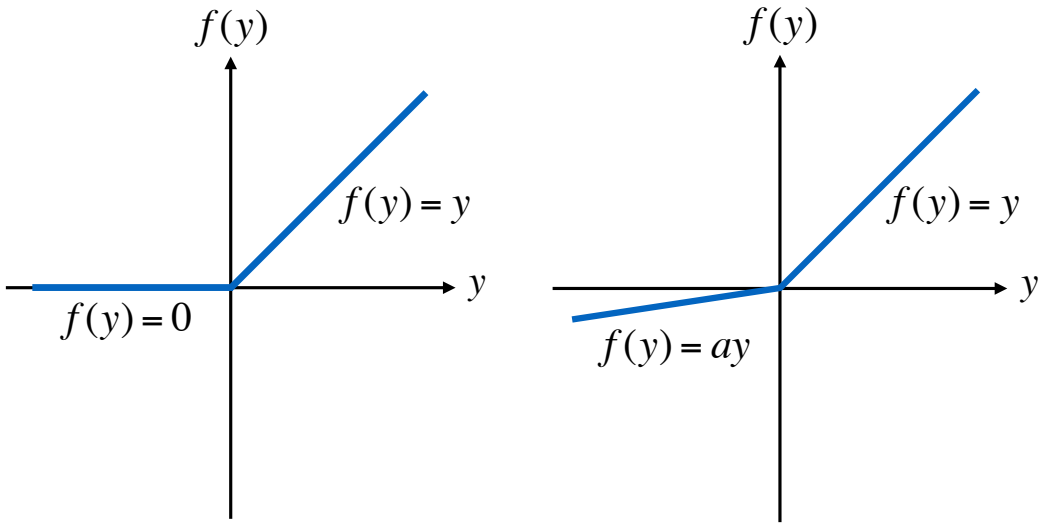
Parametric Rectified Linear Unit

The Parametric Rectified Linear Unit (PReLU) introduced by He *et al.* [21] is a modification of the ReLU. The function allows for a small non-zero gradient to flow through the output for negative input values and is defined as

$$f(x) = \begin{cases} x_i, & \text{if } x_i > 0, \\ a_i x_i, & \text{if } x_i \leq 0, \end{cases} \tag{2.10}$$

or equivalently $f(x_i) = \max(0, x_i) + a_i \min(0, x_i)$. The x_i is the input and a_i is a coefficient controlling the slope of the function output for negative input values. The subscript i on a_i shows that the parameter is allowed to be different between channels in the layer.

For small values of a the PReLU behaves as a Leaky ReLU [36] with $a = 0.01$ where the goal is to avoid zero gradients. The PReLU allows for a small 'leakage' that significantly reduce the 'dead neuron' syndrome palpable when using the ReLU. It is shown by the author that this improves the model fitting [21]. The ReLU vs. PReLU comparison is presented in Figure 2.10.



(a) ReLU (b) PReLU

Figure 2.10: The ReLU vs. PReLU activation functions. The parameter a in 2.10b is learnable [21].

One key feature of the PReLU is that the parameter controlling the output magnitude for negative inputs is a learnable parameter and is learnt jointly with the whole network. The PReLU function adds one parameter per channel in the network layers thus pose a small incremental increase to the total number of parameters in the network. This results in the network getting yet another learnable parameter to optimize with almost no extra computational cost as well as no increased risk of overfitting. This means that the network will be able to learn more specialized activation functions resulting in better performance.

2.5.3 Activation maps

A feature map is the resulting representation after the convolution filter has convolved the input signal and the result have been transformed by the activation function. The activation map contains the feature response from each convolution made by the filter in the convolution layer. Every activation map is different and determined by the weights in the convolution filter in the layer. The activation maps are stacked to form a 3D volume representation of the convolution layer filter responses.

2.5.4 Pooling layers

The pooling layer is used to alter the spatial dimension of its input.

Max-pooling

The operation reduce the number of spatial parameters for the subsequent layer input as well as increase the networks receptive field of the input. The pooling layer uses a kernel that is stepwise moved over the entire input returning only the highest pixel value within its, usually square, region and discard the rest. The result is a spatially downsized image without any interpolation being performed. The downsizing is controlled by the stride S and spatial size F of the kernel. The output dimensions from the max pooling layer can be described for an input volume $W_1 \times H_1 \times D_1$ as $W_2 = (W_1 - F)/S + 1$ and $H_2 = (H_1 - F)/S + 1$. The depth of the output stays the same as the input $D_1 = D_2$. For the pooling layer to be able to reduce the spatial size of the image the resolution of the image has to be evenly divisible by the downsampling factor. This is required since the image resolution is represented by integers. The required resolution is achieved by using zero padding at the top, bottom, left and right side of the image. Since the zeros do not affect the max pooling result it does not affect the performance of the network. An example of max pooling on a small one channel image can be seen in Fig 2.11.

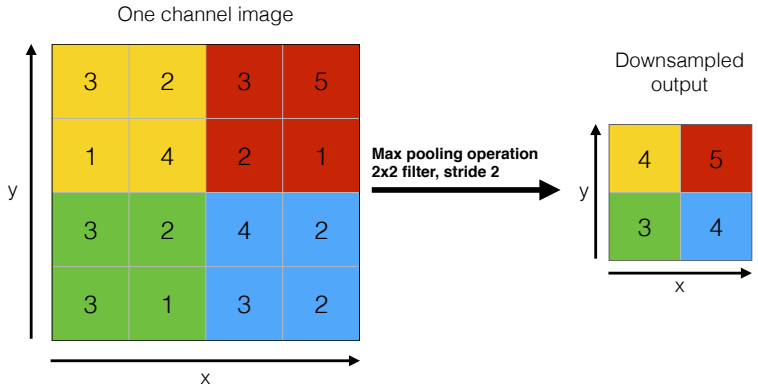


Figure 2.11: Max pooling operation with kernel size 2×2 and stride 2 on small one channel input.

Unpooling

The unpooling layer was introduced by Zeiler *et al.* [48] and is used to increase the spatial dimensions of the input. The layer operates as a reversed pooling layer, hence the name. Since the max pooling layer is non reversible an inverse reconstruction of the layer is impossible. However, by storing the indices of the maximum elements, called *switch* elements by the author, an approximation of the reconstruction can be made. The unpooling layer achieves this by using these *switch* indices to dilate the output pooling area with the maximum area, see Figure 2.12. The input data is so mapped to a spatially larger output.

2.5.5 Training improving components

Other type of layers are presented in this section that are used to improve network convergence and performance.

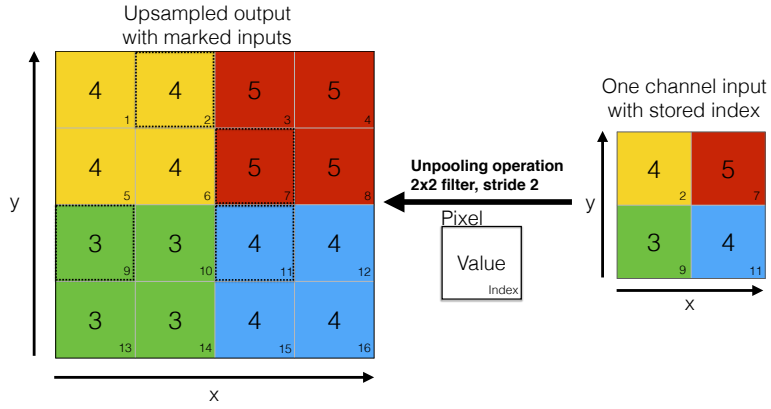


Figure 2.12: Unpooling operation on small one channel input.

Batch normalization

The Batch normalization layer was introduced by Ioffe *et al.* in 2015 [23] and it is typically applied before the activation functions in the network and drastically reduce the model training time. It increases the convergence rate, enables higher learning rates, prevents activation functions from saturating and reduce the need of Dropout as well as careful weight initialization during the network training. For non normalized input data the distribution changes between every mini-batch resulting in a covariate shift. This complicates the training procedure since the network parameters have to compensate for the shifting distribution in the input data for every mini-batch. Thus enforcing the need of a small learning rate and careful weight initialization to prevent the network from diverging during training. Batch normalization achieves this by reducing the covariate shift between mini-batches in the input to the activation functions. The algorithm utilizes a normalization step that reduce the mean to 0 and the variance to 1 for each mini-batch thus providing a more stable input. In doing so the input magnitude $|x|$ of each mini-batch is normalized resulting in a reduced risk of saturating activation functions. It was also shown that a larger batch size had a positive effect on the network performance when using batch normalization.

Spatial dropout

Spatial dropout was introduced to CNN's by [44] and is a method to reduce network overfitting. In contrary to regular dropout that set individual neuron outputs to zero, spatial dropout remove entire activation maps in the convolution layer by setting the them to zero (see Figure 2.13). That results in a different network layout for each backpropagation during the network training process. Spatial dropout acts as a learning reinforcer that forces the network to learn more robust features. This means that the network, with spatial dropout applied, cannot rely on certain activation maps in layers to always be present. It was shown by Paszke *et al.* [38] that spatial dropout increased the network performance for the semantic segmentation task so it was also implemented in the depth regression task.

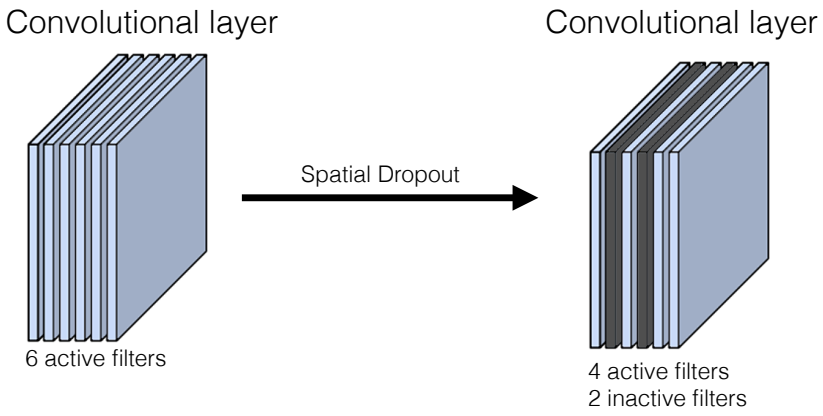


Figure 2.13: Spatial Dropout applied to a convolutional layer with six activation maps resulting in four active and two inactive activation maps as output.

2.6 Depth estimation

The network will learn to predict absolute depth in each frame by using a supervised learning scheme. A regression learning scheme will be implemented to estimate pixel-wise real valued quantities. The problem could also be posed as a classification task with a discretized class depth interval. The main advantage of this approach would be a network confidence for each pixel prediction.

2.6.1 Pixel-wise depth regression

The loss function calculates the difference between the prediction pixel and the corresponding ground truth pixel. The loss will only be evaluated in prediction pixels where ground truth is available. The local loss gradient for each neuron is then used in an optimization method to update the neuron weights. This process is repeated until there is no further improvement in the loss or the maximum number of weight updates are reached.

3. Implementation

The methods used and how they were implemented in order to estimate depth from a single monocular image with a deep convolutional neural network are discussed within this section.

3.1 ENet architecture

The efficient neural network, or ENet, introduced by Paszke *et al.* [38] is a CNN architecture designed to be computationally lightweight while maintaining high performance for tasks solved in real-time. The architecture has ∼370 000 learnable parameters requiring only 0.7 MB to be stored using 16-bit floating point precision. The network is designed with an encoder-decoder layout with bottleneck modules, first implemented in ResNet [20]. These properties results in a very fast and efficient network architecture that has low complexity and very short inference time, even on computationally weak devices. The ENet implemented on an NVIDIA TX1 embedded system was able to evaluate a 480×320 RGB input image and output a semantic segmented image at 21.1 fps. This proves that the network architecture is well suitable for solving vision problems at real time. Other semantic segmentation networks utilize many millions of parameters to solve the pixel-wise segmentation task which results in high complex models. The SegNet architecture, that is a fairly small architecture for semantic segmentation, uses 29.46 million parameters requiring 117 MB to be stored [3]. These models require much more computational resources, something that is not suitable for small low power devices. ENet compared to other CNN architectures has the highest performance per parameter measured in Top-1 accuracy for the semantic segmentation task as of March 2017 [4].

The ENet architecture utilizes all layers and features reviewed in Section 2.5 along with the architectural design features mentioned below to streamline the performance of the network.

3.1.1 Encoder-decoder

The encoder-decoder layout has proven to be an efficient way in solving pixel-wise classification problems such as semantic segmentation. ENet and other semantic segmentation networks such as SegNet [3] utilize this architectural layout. Images contain highly redundant spatial information that is reduced by downsampling the image. It proved to be an efficient way to extract and refine image features through a combination of convolution and pooling layers [38]. The encoder module downsamples the image and the decoder module upsamples the image. Image downsampling increase the receptive field of the network as well as reduce the number of computa-

tion operations in the architecture. The initial layers acts as basic feature extractors and do not directly solve the problem at hand but instead provide the consecutive layers with robust low level image features. The consecutive layers in the encoder module form high level feature representations that are sent to the decoder module where these features are refined. This provides the network with an efficient way to extract high level features from the input image.

3.1.2 Bottleneck module

The bottleneck module introduced by He *et al.* [21] allows for deeper CNN architectures without increased network complexity. This is achieved through parallelization of operations within the network. The bottleneck module in ENet consists of a main branch with pooling operations and an extension branch with convolution layers, as presented in Figure 3.1. The main branch uses either a max- or unpooling layer, depending on if the module is down- or upsampling the input. The extension branch consists of a 1×1 convolution that reduces the dimensionality, a main convolutional layer (with different kernel sizes) and a 1×1 convolution that expands the dimensionality [38]. The extension branch is element-wise added with the main branch at the bottleneck output. By introducing this parallelization, instead of a conventional pooling after convolution, allows for a 10-fold inference speed-up of the initial block. The bottleneck module construction can be seen as a decomposition of a larger convolutional filter. The filter is represented with smaller and simpler operations that could be seen as its low-rank approximation. These simpler operations also add non linearities in between them which increases the richness of the features they learn. This decomposition in the bottleneck reduce the number of parameters, maintain a large receptive field and introduce more non linearities resulting in better learned features.

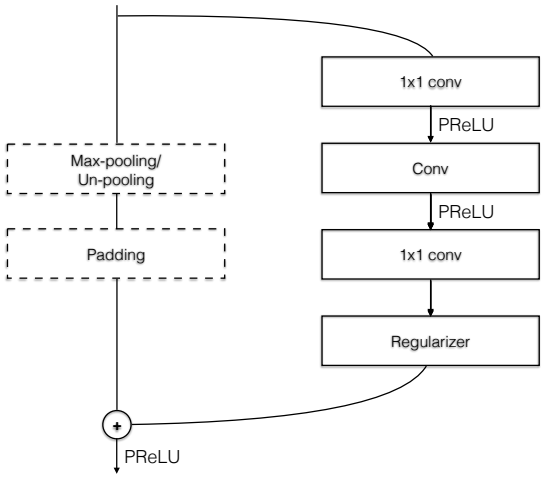


Figure 3.1: The bottleneck module used within ENet that consists of a main and an extension branch. The main branch contains a max- or unpooling operation, depending on in the module is down- or upsampling the input. The extension branch consists of convolutional layers that; reduce the dimensionality, perform regular convolution and expand the dimensionality [38].

3.1.3 Pixel-wise regression

Since ENet was designed for semantic segmentation it had to be adapted to solve the regression problem and output real valued quantities instead of performing a pixel-wise classification. This was achieved by removing the final softmax classification layer and batch normalization layer leaving the last deconvolution layer as output with one channel. The network is trained with a sparse ground truth and outputs a dense prediction without any interpolations done outside the network. The network output has the same spatial resolution as the input image. This is something that is important for the pixel-wise regression since each pixel in the output should map to each pixel in the input [24].

3.2 Network implementation tools

The CNN was implemented using the high-level API Keras [7] with TensorFlow [1] as the back-end architecture in Python. TensorFlow allows for complex data handling and neural network construction by using a tensor representation. By incorporating a tensor data type several images, or feature maps, can be stored and transferred within the network. The tensor has four dimensions where the image data with 3D spatial coordinates are stored along with the batch size of the tensor. TensorFlow provide the tools necessary to construct, train and evaluate the neural network model. The Keras API facilitates the implementation of the neural network by simplifying the commands needed to, for example, add layers to the network. All of the image handling was performed by OpenCV and all arithmetic operations were performed by Numpy.

3.3 Datasets

Two datasets were used to train and evaluate the network; the public KITTI dataset and parts of an internal Autoliv dataset. Each dataset was divided into a training, validation and test set with no overlapping sequences in-between them.

3.3.1 KITTI

The KITTI dataset [15] consists of ∼40k stereo image frames in sequences with associated LiDAR data and other telemetry data gathered over a five day period in 2011. With this dataset different algorithms for e.g. disparity image generation from stereo images or depth estimation using a single input image, could be evaluated and compared. The previously available datasets were either synthetic, did not have a depth ground truth or were captured in an indoor environment. Thus the main purpose of the dataset was to fill the void of an outdoor driving dataset containing a robust depth ground truth. The purpose was also to provide a larger quantity of data to the academia to be used as a benchmark. However, the dataset is considered relatively small compared to other image datasets used for deep learning tasks. For example, the ImageNet dataset [10] contains ∼14 million images and is constantly increasing. The dataset only contain sequences recorded during the day and in good weather conditions. This is also mentioned by the authors as a limitation and that

adding more weather and time versatile recordings to the dataset would be a future improvement.

The data was gathered by driving a vehicle with measuring equipment fitted within and around the city Karlsruhe in Germany. The data was recorded in sequences where each sequence contains one scenic recording that is comprised of several frames. The dataset is divided into 151 sequences, each assigned to one of six categories; road, residential, city, campus, person and calibration. The sensor rig contained; two stereo camera rigs; one black and white and one RGB color, one Velodyne 3D laser scanner (LiDAR) and other telemetry sensors capturing acceleration, position, speed, yaw and pitch of the vehicle. The LiDAR scanner recorded the scene depth of frame each captured by the stereo cameras and was mounted behind and in alignment with the left black and white camera.

Input images

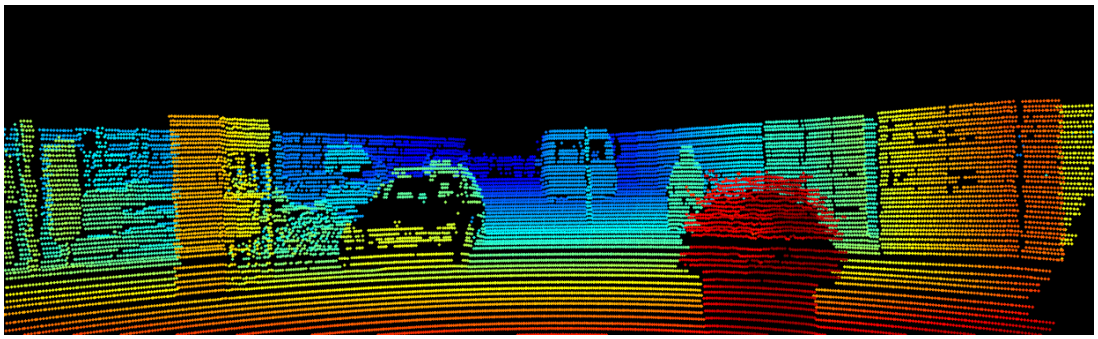
The input images used were captured by two PointGray Flea2 color cameras with 1.4 Megapixels each, mounted in a stereo rig with baseline ~ 54 cm. The provided rectified stereo image pair were individually used as input with the left camera image as primary. The position of the LiDAR scanner gave rise to a slight misalignment with the left color camera by a displacement of 6 cm. Despite the misalignment the color images were used since they provide the network with more scenic information through the provided color. The images have three 8-bit color channels that store the RGB light intensities resulting in a 24-bit image with spatial resolution between 1224×370 and 1242×375. The variation in spatial resolution only occurs in-between the days of recording and not within the sequences. This could be an effect of different camera calibrations that were conducted for each day of recording. The differences in camera calibration could possibly affect the image rectification procedure that alters the spatial resolution and warping of the images.

Ground truth

The Light Detection And Ranging (LiDAR) scanner used for ground truth depth recording in the dataset was a Velodyne HDL-64E laser scanner. The surrounding scene depth was recorded by an array of 64 vertically stacked lasers in upper and lower groups of 32 lasers each. The scanner had a 0.09 degree angular resolution and a ± 2 cm distance accuracy. The lasers covered a 360° horizontal field of view (FOV) and a 26.8° vertical FOV ($\pm 2°$ to $-24.8°$ from the horizon) with the 64 lasers evenly angularly spaced [45]. The Velodyne laser scanner also stored reflectance value from each received signal. Since these values do not affect the measured distance this parameter was not used. The point cloud from the LiDAR scanner plotted in Matlab can be observed together with the corresponding color image in Figure 3.2.

The Velodyne HDL-64E has a 50 meter range for pavement and a 120 meter range for cars[1] [45]. The same observations of maximum depth in the dataset were made as Godard *et al.* [18] thus the maximum depth was set to 80 meters. The rotation of the LiDAR scanner over the camera FOV is described by three time stamps in the dataset. The time stamps are distributed over the camera FOV. The

[1]The provided maximum depth specified in [16] and [15] were 100 meter and 120 meter for the dataset they collected themselves. According to [18] the maximum measured depth in the KITTI dataset was 80 m.



(a) LiDAR depth map.

(b) The input image.

Figure 3.2: The point cloud from the LiDAR scanner visualized in Matlab using the provided developers kit [15]. Large points with a colormap for each depth measurement for visualization purpose in Figure 3.2a together with the corresponding input image in Figure 3.2b.

first and last time stamps were recorded at the horizontal beginning and end of the FOV. The middle time stamp was recorded when the LiDAR scanner coincided with the center of the camera FOV. The image was captured, along with other sensor readings of GPS and telemetry, at the middle time stamp when the LiDAR scanner triggered a reed sensor mounted on it.

The number of points per horizontal degree (25000 points / 360°) are much larger than the number of points per vertical degree (64 points / 26.8°) [45]. This results in the recorded point cloud having a higher horizontal than vertical point density. Thus the recorded LiDAR data produces a point cloud with 64 horizontal dotted lines with 0.09 degree vertical angular displacement containing the scenic depth information. The data points in the ground truth only maps to $\sim 5\%$ of the pixels in the RGB input image hence resulting a sparse ground truth depth map.

Image preprocessing

The input images in the KITTI dataset had a varying spatial dimensions between frames, that created a problem since the network required a fixed input spatial resolution. This was solved by cropping all the images to 1224×370, the lowest resolution found in the dataset. This was carried out in MATLAB by including pixels from indices $(0, 0)$ to $(370, 1224)$ while discarding the rest. The images were

exported as 24-bit RGB color `.png` image files.

Depth data extraction

The depth data from the KITTI dataset is stored in `.bin` files and was extracted using MATLAB. The provided developers kit was modified from plotting individual points to a projection of the measured points to an empty array with same spatial dimensions as the camera image. Each laser point measurement was stored in one pixel in the new image. The projected depth data was then stored as a `.png` image file. To keep high precision in the exported depth data the image file was stored using a 16-bit data type resulting in an discretization interval $I = [0, 65535]$. The depth discretization with this scaling would be $\frac{80\text{meters}}{2^{16}-1} = 0.0012$ meters which is good enough since the precision of the LiDAR scanner is 0.02 meters. Before storing the depth measurements of the `.bin` file in the 16-bit `.png` file the values were normalized at the maximum measured depth of 80 meter and scaled by $(2^{16} - 1)$. The resulting ground truth image is presented in Figure 3.3. Note that the LiDAR scanner construction causes the ground truth data to only cover the lower half of the image and is very sparse.

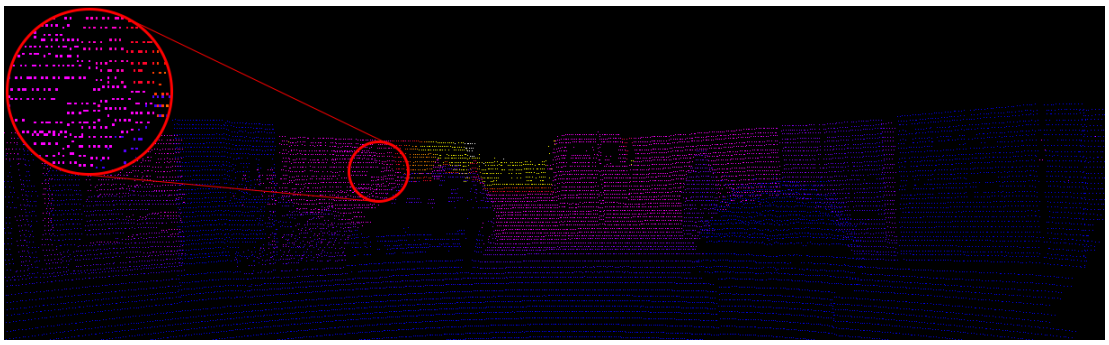


Figure 3.3: The extracted sparse ground truth from the KITTI dataset [15]. The intensity of each pixel correspond to the measured depth on that location. Note that the file used in the network training is a 16-bit single channel `.png` file.

3.3.2 Dataset split

The dataset was split into training, validation and test sets. The split of the KITTI dataset has been an issue in the literature since no official dataset split was provided by Geiger *et al.* [15]. Eigen *et al.* [13] proposed a split of the KITTI dataset which has been the standard in the literature. They provided a test set file list along with their paper but did not provide any training and validation set file lists. This test set file list have been used as the benchmark between depth estimation models in literature. However, since no training and validation sets have been available no valid comparison between models have been possible[2]. Different methods of

[2]Godard *et al.* [18] updated their paper with a set of file lists for the training and validation sets in April 2017, which was after this split was performed. These file lists were used to compare the proposed model to the literature to enable true model comparability. However, the model development had progressed and model training settings were chosen from parameter searches that were performed using the proposed split.

removing frames from the sequences in the test set have previously been proposed by [18, 14, 27]. However, these frame removing methods result in an inconsistent total number of frames between papers and also do not refer to which sequences were used in the training set. Another problem is the depletion of stationery frames in the training set. Enough information about this is also not provided in literature. This results in all papers previously reporting different sizes of the training set which affects the model training and the network performance differently. However, Godard *et al.* [18] updated their paper with complete file lists of the test, training and validation sets in April 2017. This made it possible to exactly compare methods which facilitated the performance evaluation of the model greatly.

Another split of the KITTI dataset was also performed. The test set proposed by Eigen is a very small set of only 697 images. Therefor another split is proposed with a larger test set, consisting of 50% of the images in the dataset, with categories road, residential and city. The purpose of this split was primarily the unavailability of an official split and to evaluate the trained network on more test frames, to study the generalization of the model.

Eigen split with Godard file list

The KITTI dataset split performed by Eigen *et al.* [13] is the split used as benchmark in the literature. Eigen proposed a split of 28 scenes for training and 28 for test which has been the standard in the literature. The categories used from KITTI were; road, residential and city and contained a total of 61 sequences where only 56 were used. The other sequences within the three categories contained stationary frames and were therefore removed. The file lists[3] provided by Godard *et al.* [18] contain 22 600 training frames, 888 validation frames and the 697 test frames, proposed by Eigen.

Balanced split

Since the KITTI dataset is relatively small (∼40 000 frames from each camera) a careful split into non-overlapping training and test set was performed. The dataset was split into 28 training scenes and 28 test scenes proposed by [13] with 50% training and 50% test. The split resulted in 20 360 frames in the training set and 20 131 frames in the test set[4]. The validation set frames were selected as a subset of the training set with non overlapping sequences. The other categories in the dataset contained stationary recordings and were not used in the project. To ensure no overlapping sequences in the training and test set a rigorous supervision of scene differences was performed while splitting the dataset. This was enforced by selecting sequences that were separated by their caption time and recording environment. The dataset also contains a number of frames within some sequences where the vehicle was stationary. These frames could cause the network to overtrain on those specific scenes which would negatively affect the model performance and are thus unwanted in the training set. The removal of stationary frames was performed by visual inspection of all training scenes to locate and remove frames where the vehicle was stationary. A validation set was created from a ∼20% subset of the training set. The sequences in the validation set were chosen with the same caution used

[3]Eigen KITTI split file lists available at `https://github.com/mrharicot/monodepth`

[4]Balanced KITTI split file lists provided at `https://github.com/jschennings/mono_depth`

when selecting the test set. This would ensure the uniqueness of scenes so that the network could be validated on unique previously for it not seen frames.

3.3.3 Autoliv

The Autoliv dataset consists of a very large collection of stereoscopic video recordings. The stereo images were recorded in different countries during different times of day with different lighting condition and weather scenarios. These data collections do not contain any LiDAR ground truth data so the ground truth data had to be constructed using a stereo disparity algorithm. The dataset is used to examine the networks capability of being trained on another type of data with different input images and ground truth type.

Input images

The input image used in the network training was one of the images from the stereo camera. The images from the camera were rectified before the network training and was done in order to transform the images onto the same image plane. The images also contained several channels of color information.

Ground truth images

This disparity image from the stereo pair is used as the ground truth in the network training. The disparity image for each frame pair is calculated by the stereo disparity algorithm used in Autoliv's commercial products. The depth information in the disparity image is calculated by measuring the displacement of an objects pixels between the left and right image of the stereo camera. The calculated disparity is then stored as an intensity in the image. The disparity is inversely proportional to the depth, $Z(\mathbf{x})^{-1}$ at pixel $\mathbf{x}$ in the disparity image.

Dataset split

The Autoliv dataset split was performed by choosing different sequences from different days and different locations. A subset of $\sim 20\%$ of the training set was used as validation set. The network was evaluated on a test sequence of previously unseen frames.

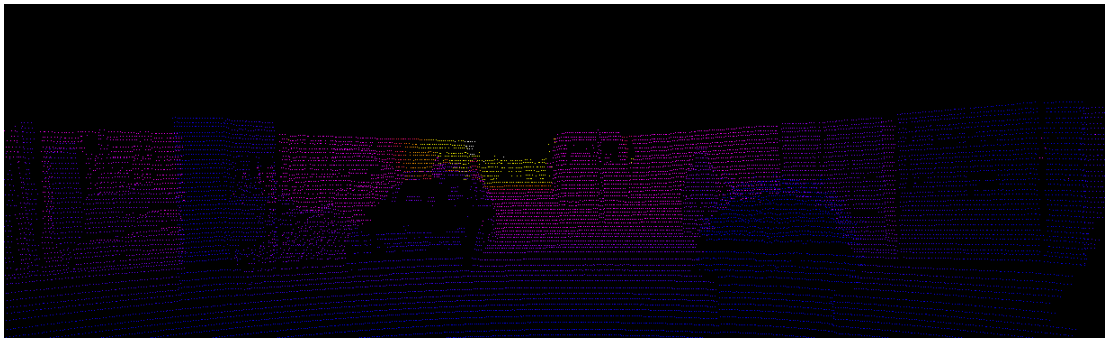
3.4 Network training

The steps taken in the network training process are discussed within this section. The network was initially trained using weighted convolved dense ground truth depth maps as an initial sanity check. The network was then trained on the sparse ground truth where a mask function was introduced in the loss function to remove unwanted pixels.

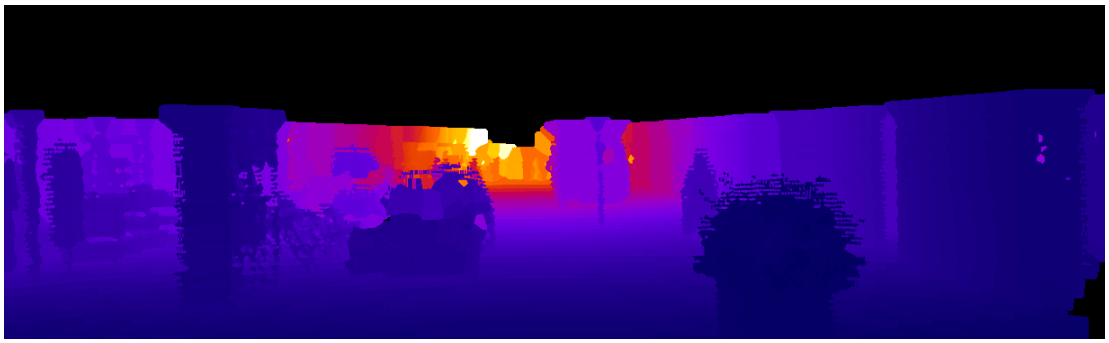
3.4.1 Dense ground truth

As an initial sanity check of the network's capability in solving the regression problem a dense ground truth was used in the network training. The initial training

was performed on the dense ground truth since it did not require any masking of the non-valid pixels present in the sparse ground truth. The dense ground truth was constructed from the `.png` sparse ground truth imaged extracted from the `.bin` data files, described in Section 3.3.1. A weighted convolution was performed in MATLAB on the sparse ground truth files to produce the dense ground truth representation, presented in Figure 3.4. The resulting dense ground truth images were not aesthetically perfect but fulfilled the task and were used in the network training.



(a) Sparse ground truth



(b) Dense ground truth

Figure 3.4: The dense representation (a) of the sparse ground truth (b) constructed with a weighted convolution in MATLAB.

3.4.2 Sparse ground truth

To enable the network training on a sparse ground truth image the pixels without depth information were to be left out from the loss evaluation. The network solves a regression problem that demands each pixel in the prediction evaluated in the loss function has to have a corresponding spatial pixel in the ground truth image. Pixels in the prediction without corresponding pixels in the ground truth are referred to as non-valid pixels. If these non-valid pixels were evaluated in the loss function the network would learn features corresponding to depthless areas in the ground truth image. Therefore a mask function was added to the loss function to remove the non-valid pixels from the loss evaluation in each frame. By removing the non-valid pixels from the learning process the network would be able to output a spatially coherent dense depth prediction image.

3.4.3 Mask function

The mask function remove pixels from each sample prediction without ground truth depth information during the network training. The mask is a binary version of the ground truth image and has coinciding spatial dimensions. Only the pixels with depth values in the ground truth have corresponding pixels in the binary image that are valid, or set to one. The mask was created by first storing all indices for the non-zero pixels in the ground truth image and then set the corresponding pixels in the mask to one. The network prediction was then multiplied with the mask, efficiently removing pixels without ground truth, to not include these in the loss function. The mask function also posed no problem in the differentiation of the loss function.

3.4.4 Regularization term

To address the issue of spatially incoherent data in the sparse ground truth a regularization term was added to the loss function. A dense ground truth image provides the network with spatially coherent data and assists the network to predict a dense continuous output. However, sparse ground truth data contains spatially incoherent data. The sparse ground truth is highly discontinuous with large gradients over the image. To aid the network in predicting a continuous and spatially coherent dense depth map a regularization term was added to penalize large image gradients. The regularization term is defined by

$$\lambda \sum_{i=1}^{n} [(\nabla_x d_i)^2 + (\nabla_y d_i)^2] \tag{3.1}$$

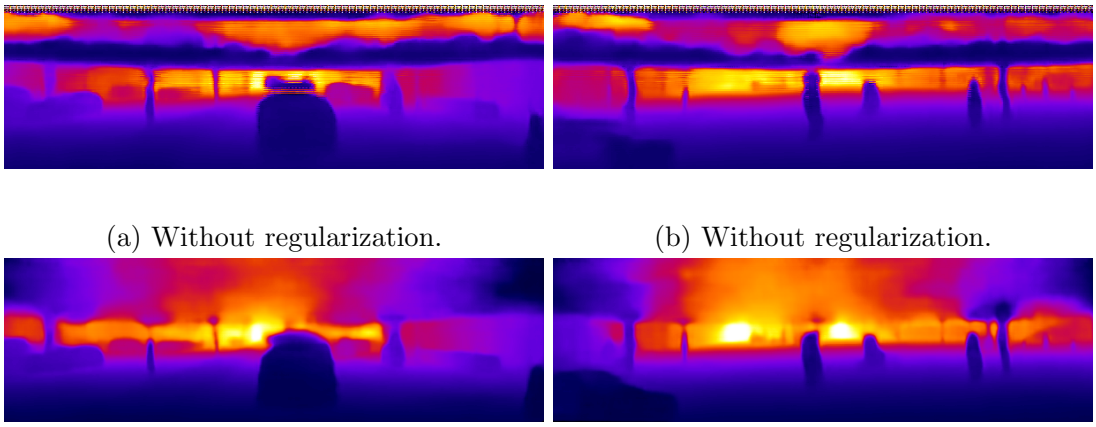
where n is the number of valid pixels in the frame, λ is the regularization parameter that determines the magnitude of the regularization. The $d_i = |\rho(x_i) - Z(x_i)|$ where $\rho(x_i)$ is the network prediction and $Z(x_i)$ is the ground truth at pixel x_i. The ∇_x and ∇_y are the horizontal and vertical image gradients and were calculated by a convolution of the error with the kernels

$$\begin{aligned} \nabla_x d &= \begin{bmatrix} 1 & -1 \end{bmatrix} * d \\ \nabla_y d &= \begin{bmatrix} 1 \\ -1 \end{bmatrix} * d. \end{aligned} \tag{3.2}$$

The regularization term results in a high penalty for strong image gradients thus enforcing the network to predict a continuous depth map with spatially coherent data. The results of using the regularization in the network loss function are presented in Figure 3.5.

3.4.5 Optimization method

The optimization method in a neural network minimize the gradient of the loss function with respect to the weights of the network prediction. The Adam optimization method introduced by Kingma *et al.* [25] was the optimization method used when training the ENet architecture. The parameters within the Adam optimizer were constant and set to; $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e - 8$ and learning rate 10^{-4}. The



(a) Without regularization. (b) Without regularization.

(c) With regularization $\lambda = 0.1$. (d) With regularization $\lambda = 0.1$.

Figure 3.5: The network prediction without regularization in 3.5a and 3.5b and with regularization factor $\lambda = 0.1$ in 3.5c and 3.5d.

parameters β_1, β_2 and ϵ are internal optimization parameters. The learning rate is used to control the the magnitude of the network weight update in each iteration to control the rate of the optimization. The Adam optimizer also features a learning rate decay option which allows for an incremental decay of the learning rate for each iteration. The learning rate decay was set to 0 as by default.

3.5 Evaluation metrics

To evaluate the performance of each trained network a set of error evaluation metrics, proposed by Eigen *et al.* [13], were used. These are the standard evaluation metrics used in the literature to evaluate depth estimation model performance. These metrics were designed to measure prediction error on different scales to cover all aspects of possible global and local error. The error in each prediction was evaluated towards the corresponding ground truth image in pixels where depth information were available. The used error evaluation metrics were; the root mean square error (RMSE), RMSE of the logarithm of the error, absolute relative difference (ARD), squared relative difference (SRD) and the prediction accuracy,

$$
\begin{aligned}
\text{RMSE: } & \sqrt{\frac{1}{T}\sum_{i=1}^{T} ||\rho(\mathbf{x}_i) - Z(\mathbf{x}_i)||_2^2}, \\
\text{RMSE log: } & \sqrt{\frac{1}{T}\sum_{i=1}^{T} ||\log(\rho(\mathbf{x}_i)) - \log(Z(\mathbf{x}_i))||_2^2}, \\
\text{ARD: } & \frac{1}{T}\sum_{i=1}^{T} \frac{|\rho(\mathbf{x}_i) - Z(\mathbf{x}_i)|}{Z(\mathbf{x}_i)}, \\
\text{SRD: } & \frac{1}{T}\sum_{i=1}^{T} \frac{|\rho(\mathbf{x}_i) - Z(\mathbf{x}_i)|^2}{Z(\mathbf{x}_i)}, \\
\text{Accuracy: } & \frac{1}{T}\sum_{i=1}^{T} \left[\max\left(\frac{\rho(\mathbf{x}_i)}{Z(\mathbf{x_i})}, \frac{Z(\mathbf{x_i})}{\rho(\mathbf{x}_i)} \right) < thr_j \right], \quad thr = [1.25, 1.25^2, 1.25^3]
\end{aligned}
\tag{3.3}
$$

where $\rho(x_i)$ is the predicted depth and $Z(x_i)$ is the ground truth at pixel x_i. T is the number of pixels with depth data in the ground truth image. The accuracy measurement, where [ ] are the Iverson brackets, divides the pixels into error intervals governed by the threshold values. The error measurement of each image is summed up and then the average for the whole evaluation set is derived. Eigen *et al.* [13] proposed these values that correspond to a prediction accuracy of $\{\pm 25\%, \pm 56.25\%, \pm 95.31\%\}$ from the ground truth depth.

3.6 Accuracy visualization

The network accuracy visualization method highlights the pixels within the network prediction that lies within a given accuracy interval from the ground truth. In doing so the prediction accuracy is coupled with the spatial information within the image to present a more intelligible view of the network prediction. This provides important visual information about where in the image the network makes capable predictions, something that is not captured by quantitative studies. The proposed accuracy threshold values provide a broad and inexact interval for the network prediction with the lowest precision level at $\pm 25\%$. This accuracy was not sufficiently high for the task at hand which demanded a higher accuracy of at least $\pm 10\%$. The threshold intervals were extended with $\pm 5\%$ and $\pm 10\%$ intervals together with a final term collecting pixels with error outside the interval. The proposed accuracy visualization method presents the pixels in each confidence group with a separate color.

Since the pixels within the highest accuracy interval are a subset of the pixels within the worst accuracy interval each group was individually marked so that pixels in $\Omega_{\pm 5\%} \notin \Omega_{\pm 10\%}$. The accuracy visualization in Figure 3.6 clearly show in which parts of the image the network is able to make good prediction and were it fails to do so.

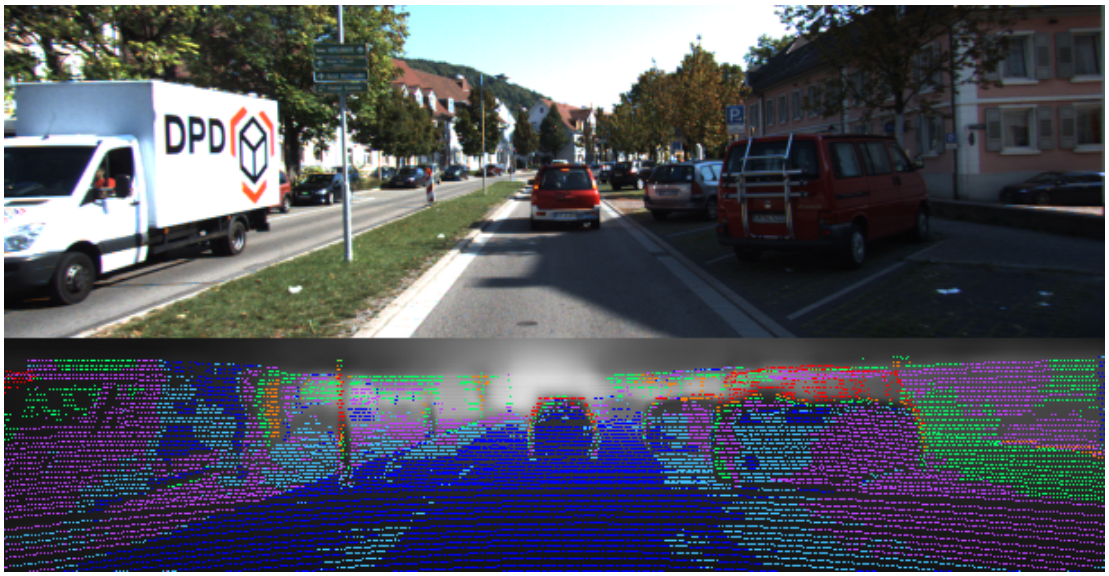


Figure 3.6: Accuracy visualization of the predicted image together with corresponding input frame. Each color corresponds to one error interval.

3.7 Parameter optimization

The optimization problem contains many hyperparameters that needs to be optimized to increase the network performance. A parameter search was performed in order to optimize two hyperparameters, the loss function $\mathcal{L}$ and regularization parameter λ. The examined loss functions were; the $\mathcal{L}_1$ norm, the $\mathcal{L}_2^2$ norm and the berHu norm, all defined in Section 2.3.1. The regularization parameter λ was evaluated for the values 0, 0.1 and 1. The values were chosen to examine how the network prediction was affected by changes in order of magnitude for λ.

3.8 Data dependency analysis

In order to evaluate the data dependency of the regression problem the network was trained for a fixed number of iterations, or weight updates, with a reduced training set. To provide an equal comparison between the entire training set and the reduced training set subsets the total number of weight updates had to be equal. The exact number of training samples used is determined by the training set and the batch size by modulo(training set, batch size) = 0. The entire training set with the Eigen split [13] and Godard *et al.* [18] file lists and a batch size of 6 results in 22 596 training frames. The number of weight updates were set to 903 480, corresponding to a 40 epochs training on the resulting entire training set. Different subsets Ω_i, where $i = \{1680, 3360, 8070, 11298, 16140\}$ denotes the number of frames, were selected from the entire training set. The subset sizes were chosen to reduce the weight update difference to zero between the subsets and the entire training set. The frames were randomly selected without any consideration taken to sequence number from where the frames were selected.

3.9 Input file variation studies

The methods within this section were implemented with the ambition to possibly improve the prediction performance of the network.

3.9.1 Previous frames

Previous frames were added to the network in order to extend each training sample to provide the network with more scenic information. In doing so the network would be presented with information from each scene to ideally learn features related to structure from motion. The previous frame was added by expanding the input channels to fit the wanted number of previous frames.

3.9.2 Right frame

The training set was extended with the images captured by the right stereo camera in order to provide the network with more scenic information. The network would use the ground truth image and the left or right image separately in the network training. The right images were added to possibly improve the network training.

3.9.3 Data augmentation

Data augmentation was applied on the training set in order to provide the network with additional training data. The data augmentation purpose is to alter the image data so that the augmented image is presented to the network as a new image. The data augmentation can alter the pixel values by changing brightness, contrast and gamma, cropping, resize and rescaling. The data augmentation used in the network was a horizontal flip.

Horizontal flip

A horizontal flip data augmentation was applied to the training set during the network training. The probability of a horizontal flip was set to 50% for each training sample during the network training.

3.10 Network generalization

The networks generalization capability was evaluated by training the network on the KITTI dataset with the Eigen split and evaluating it on the Cityscapes dataset. Since the Cityscapes dataset does not contain a depth ground truth the network predictions will only be qualitatively analyzed.

Cityscapes

The dataset Cityscapes was created by Cordts *et al.* [8] for semantic segmentation and provides a stereo dataset with annotation data. The images are collected from several German villages and construct a more generalized dataset than the KITTI dataset. The dataset images have different spatial resolution, aspect ratio and focal length compared to the images in the KITTI dataset. The images are also captured

with a slightly different point of view. The Cityscapes dataset is used to evaluate how the network performs for a different set of images than what it has been trained on.

In order to evaluate the images in the network they were spatially resized and cropped to the same spatial size as the KITTI input images. Since the aspect ratio of the Cityscapes images were different equal parts of the top and bottom of the image were cropped, saving the vertical center image.

4. Results

The quantitative and qualitative results from the network training are presented in this section. The network was trained; on the KITTI dataset using an NVIDIA GTX 1060 6GB GPU and on the Autoliv dataset a NVIDIA GTX Titan X 12GB GPU. The default network training settings for the KITTI network were; the Eigen split, 40 epochs, downsampling factor = 2, the masked $\mathcal{L}_1$ norm, learning rate 0.001, batch size 6 and horizontal flip probability 50%. These training settings were chosen since these resulted in a training time of roughly 12 hours on the balanced split and 24 hours on the Eigen split. This facilitated the network development and was enough time for the network to converge and produce good results.

4.1 KITTI dataset results

The qualitative and quantitative results from the network training and evaluation on the KITTI dataset using the split proposed by Eigen *et al.* [13] and file lists from Godard *et al.* [18]. The split contained 22 600 training frames, 888 validation frames and 697 test frames. With the maximum batch size of 6 samples possible on the NVIDIA GTX 1060 6GB GPU the total number of training frames were 22 598. The network was evaluated on the test set unless other is stated.

4.1.1 Quantitative results

The quantitative results on the KITTI dataset with the Eigen split. The network training settings in addition to the default settings were 100 epochs, regularization parameter $\lambda = 0$ and $\mathcal{L}_2^2$ norm as loss function. The resulting settings produced 2 259 800 weight updates during the network training. The training time was 52.3 hours. The quantitative results are observed in Table 4.1.

4.1.2 Qualitative results

The qualitative evaluation of the network training on the KITTI dataset. The network was evaluated on each test set. The network training settings for the Eigen split in addition to the default settings were; 100 epochs and regularization parameter $\lambda = 1$. The qualitative results are presented in Figure 4.1. Results of the network not performing well is presented in Figure 4.2.

[1]Results from Godard *et al.* [18].

[2]Results from the network training on the Kitti dataset alone.

[3]Do not use the exact same training set as Godard *et al.* but same test set, so comparison is not truly fair.

Table 4.1: Quantitative results of network performance on the test set of the KITTI dataset split proposed by Eigen et al. [13]. The results are taken from the paper by Godard et al. [18]. Their results are from the network training on only the KITTI dataset. The results from Eigen et al. and Laina et al. are taken from the Godard paper [18]. The results from Kuznietsov et al. [27] were produced by network training on their own dataset split inspired by the Eigen split and use the Eigen test set and are therefore not truly comparable.

	RMSE	RMSE log	ARD	SRD	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Methods	Lower is better				Higher is better		
Train mean set[1]	8.102	0.377	0.361	4.826	0.638	0.804	0.894
Eigen *et al.* [13]	6.307	0.282	0.203	1.548	0.702	0.890	0.958
Liu *et al.* [34]	6.471	0.273	0.201	1.584	0.680	0.898	0.967
Godard *et al.* [18][2]	5.927	0.247	0.148	1.344	0.803	0.922	0.964
Zhou *et al.* [49][2]	6.856	0.283	0.208	1.768	0.678	0.885	0.957
Kuznietsov *et al.* [27][3]	4.621	0.189	0.113	0.741	0.862	0.960	0.986
Proposed	4.915	0.199	0.124	0.963	0.847	0.950	0.980

4.2 Parameter search

The result from the parameter search are presented in Table 4.2. Each parameter setting was trained with the default training settings.

Table 4.2: Quantitative results of network performance on the validation set from the parameter search performed on the KITTI dataset with Eigen *et al.* split [13] and Godard *et al.* [18] file lists. The network was trained with the default training settings. Training time was on average 24 hours. Best results marked in bold.

		RMSE	RMSE log	ARD	SRD	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Loss	λ	Lower is better				Higher is better		
berHu	0	3.237	0.139	0.075	0.470	0.932	0.978	0.991
norm	0.1	3.373	0.146	0.084	0.532	0.922	0.976	0.990
	1	3.550	0.150	0.085	0.543	0.916	0.974	0.990
Mean	0	3.202	0.139	0.074	0.469	0.933	0.978	0.991
absolute	0.1	3.240	0.138	0.075	0.477	0.933	0.979	0.991
error	1	3.282	0.139	0.076	0.473	0.930	0.978	0.991
Mean	0	**3.148**	**0.135**	**0.072**	**0.442**	**0.936**	**0.979**	**0.991**
squared	0.1	3.224	0.139	0.078	0.479	0.929	0.978	0.991
error	1	3.269	0.139	0.076	0.476	0.931	0.978	0.991

The prediction error plots for each network training with the different parameter settings are presented in Figure 4.6 and 4.7.

4.3 Reduced dataset

The quantitative and qualitative results from the network training on a reduced training set. The default training settings were used with the Eigen split and Godard file lists and regularization parameter $\lambda = 0$.

The quantitative results from the training on the reduced dataset are presented in Table 4.3. The network training results are also visualized in Figure 4.3. Qualitative results for the different training sets for one frame are presented in Figure 4.4.

Table 4.3: Quantitative results from the reduced training set on the Eigen split [13] on the KITTI dataset. The number of weight updates were fixed at 903 840 and the samples were chosen randomly from the training set with no regard taken to sequence distribution.

Subset frames	1680	3360	8070	11298	16140	22596	
% of training set	7.4	14.9	35.7	50	71.4	100	
Number of epochs	538	269	112	80	56	40	
RMSE	3.894	3.517	3.228	3.261	3.190	**3.167**	Lower is
RMSE log	0.169	0.153	0.140	0.141	0.138	**0.138**	better
ARD	0.099	0.088	0.075	0.077	**0.073**	0.076	
SRD	0.698	0.596	0.474	0.489	**0.454**	0.474	
$thr < 1.25$	0.893	0.915	0.931	0.931	**0.934**	0.934	Higher is
$thr < 1.25^2$	0.965	0.972	0.977	0.977	0.979	**0.979**	better
$thr < 1.25^3$	0.986	0.988	0.990	0.990	0.991	**0.991**	

4.4 Input data variation studies results

Results from the various configurations used on the samples in the network training. The network was trained on the Generalization dataset split with the default training settings and regularization parameter $\lambda = 1$ and evaluated on the validation set. Only one previous frame was tested in the network training since this sample setting performed slightly worse than the network trained on the regular frame. The right frame sample configuration had twice the amount of training samples since the right frame also is used in the training set and was only trained for 20 epochs to preserve the number of weight updates. The network trained on only one input sample without any modifications is referred to as the regular frame.

Table 4.4: The quantitative results from the various sample configurations where different modifications to the input sample were evaluated. All networks were trained on the default training settings with the generalization dataset split, $\lambda = 1$ and evaluated on the validation set.

	Regular sample	One previous frame	Right frame	Horizontal flip	
RMSE	5.985	6.993	6.130	**5.880**	Lower is
RMSE log	0.252	0.284	0.264	**0.250**	better
ARD	0.155	0.178	0.166	**0.150**	
SRD	1.370	1.993	1.523	**1.261**	
$\delta < 1.25$	0.773	0.744	0.759	**0.775**	Higher is
$\delta < 1.25$	**0.917**	0.894	0.904	0.916	better
$\delta < 1.25$	**0.966**	0.953	0.961	0.966	

The qualitative results are compared to the prediction made by the network trained using the regular sample and presented in Figure 4.5.

4.4.1 Previous frames

Qualitative results are presented in Figure 4.5g, 4.5h and 4.5i.

4.4.2 Right frame

Qualitative results from adding the right frame to the network training are presented in Figure 4.5j, 4.5k and 4.5l.

4.4.3 Horizontal flip

The qualitative results from the horizontal flip data augmentation are presented in Figure 4.5m, 4.5n and 4.5o.

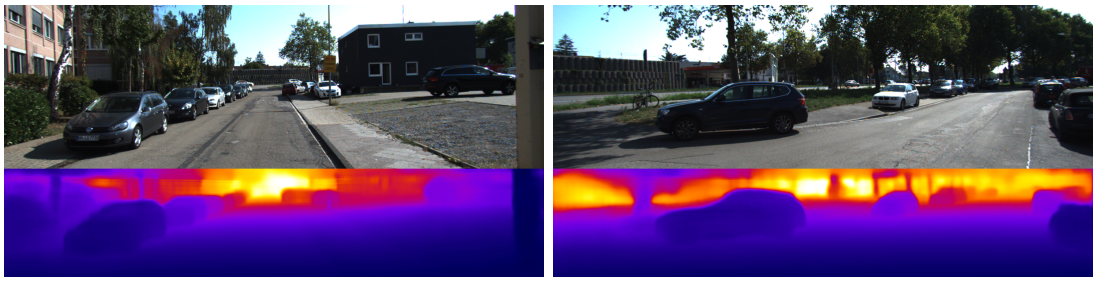
4.5 Autoliv dataset evaluation

The qualitative results from the network trained on the Autoliv dataset using a disparity image as ground truth. No changes were made to the network architecture in order to train on this dataset, only input and data handling was modified. The network was trained for 2 epochs with ∼500 000 training frames. Some of the frames where the network performs well in predicting the depth are presented in Figure 4.8. Some examples of the network making a bad depth prediction are presented in Figure 4.9.

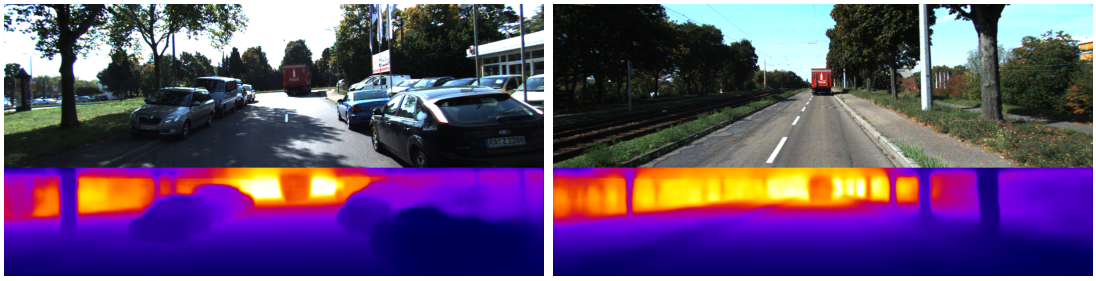
4.6 Generalization to Cityscapes

The results of the generalization capability of the network trained on the KITTI dataset evaluated on frames from the Cityscapes dataset. The network was trained for 100 epochs with the Eigen split and Godard file lists with default training settings, $\lambda = 0$ and $\mathcal{L}_2^2$ norm as loss function. Qualitative results are presented in Figure 4.10.

Figure 4.1: Qualitative results from the KITTI dataset. Images are best presented on a digital medium.

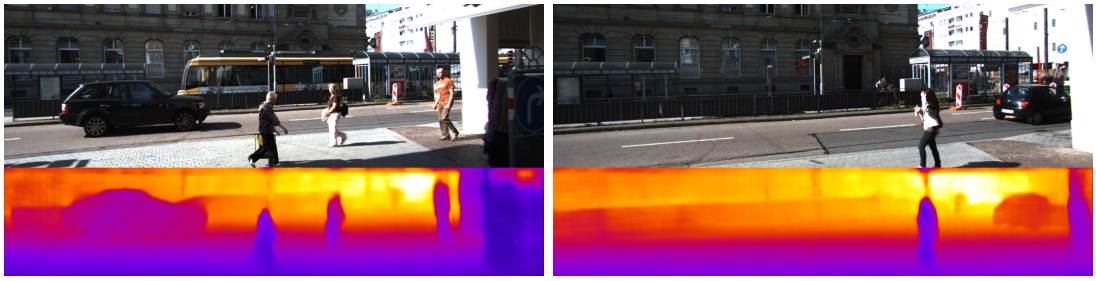


(a) (b)



(c) (d)

(e) (f)



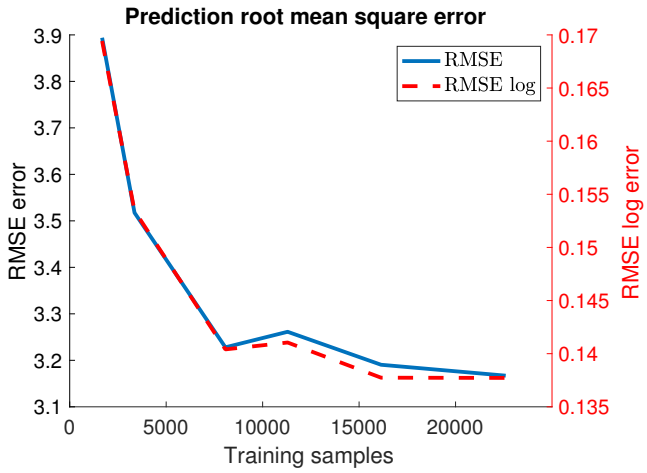
(g) (h)

Figure 4.2: Bad qualitative results from the KITTI dataset. The images are best presented on a digital medium.

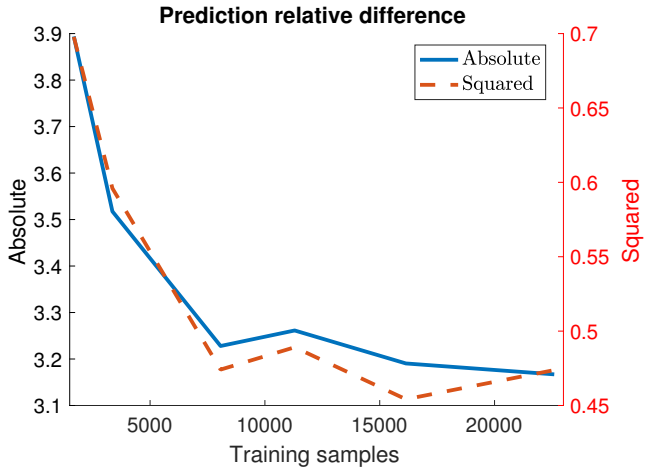
(a) Depth of pedestrian not registered in frame.

(b) Missing pedestrians and also erroneous scaling.

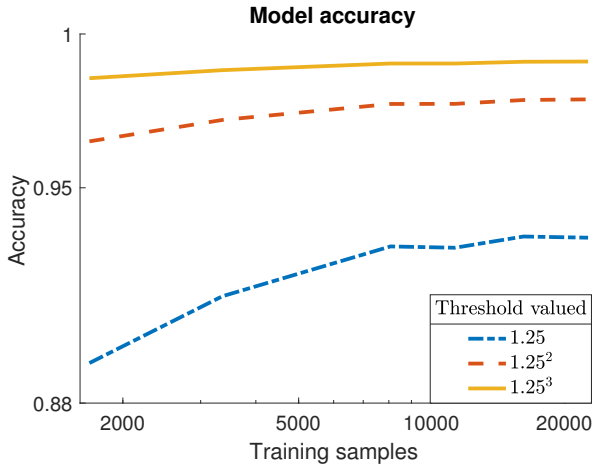
Figure 4.3: The error evaluation metrics for the reduced training set.



(a) RMSE, lower is better



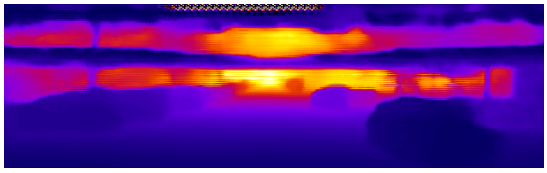
(b) Relative difference, lower is better



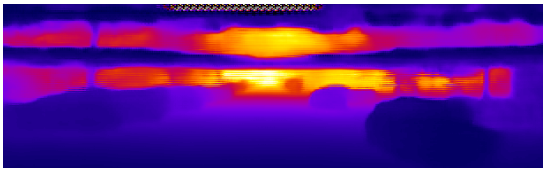
(c) Accuracy, higher is better

Figure 4.4: The quantitative results from the network training on the reduced datasets. Images are best presented on a digital medium.

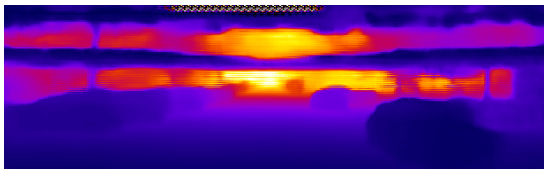
(a) The input image



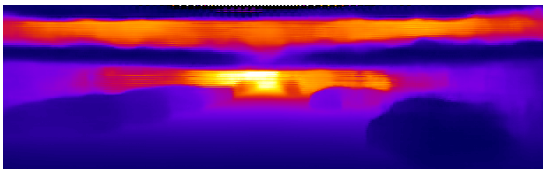
(b) Subset Ω_{1680}



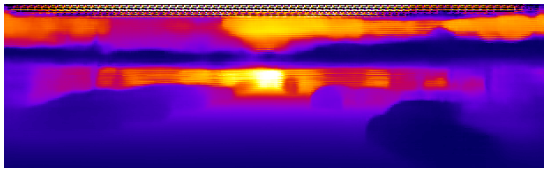
(c) Subset Ω_{3360}



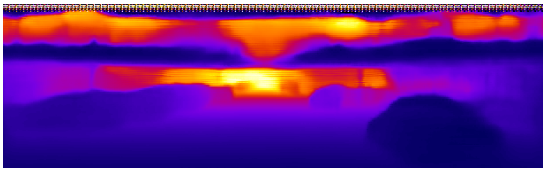
(d) Subset Ω_{8070}



(e) Subset Ω_{11298}



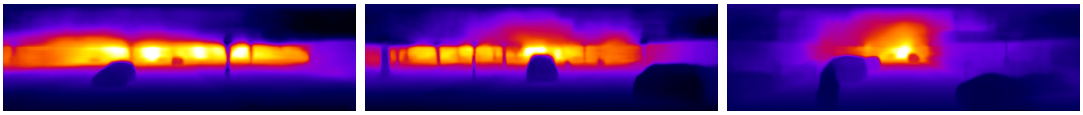
(f) Subset Ω_{16140}



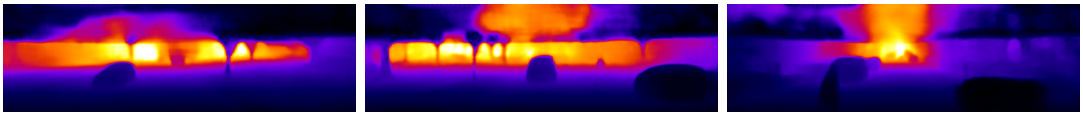
(g) Subset Ω_{22600}

Figure 4.5: The qualitative results from the input data variation studies with different sample settings. Images are best presented on a digital medium.

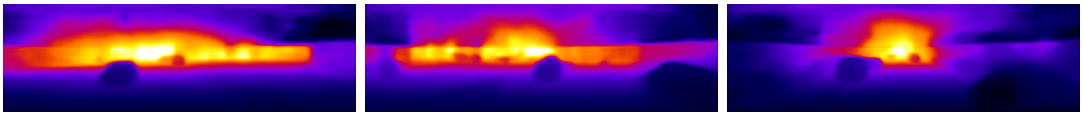
(a) Input frame. (b) Input frame. (c) Input frame.



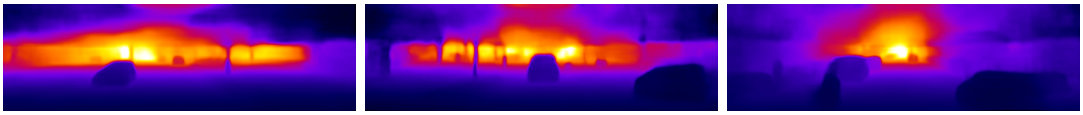
(d) Regular frame. (e) Regular frame. (f) Regular frame.



(g) One previous frame. (h) One previous frame. (i) One previous frame.

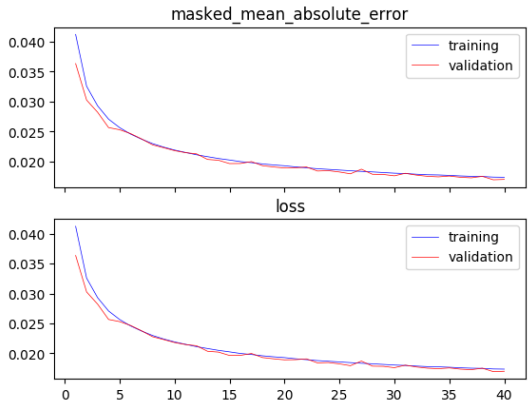


(j) Right frame. (k) Right frame. (l) Right frame.

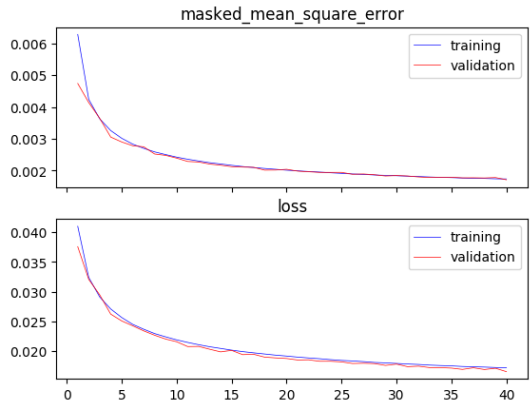


(m) Horizontal flip frame. (n) Horizontal flip frame. (o) Horizontal flip frame.

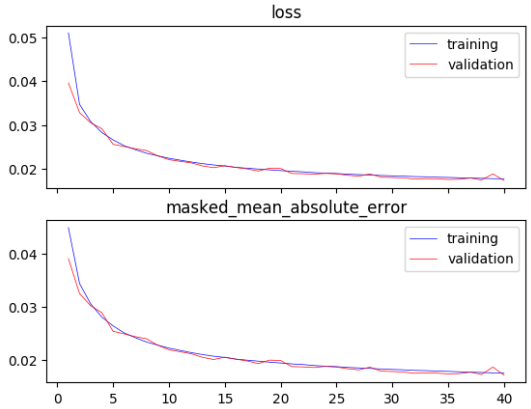
Figure 4.6: Plots for the loss of the mean absolute error and mean square error norm with $\alpha = [0, 0.1, 1]$ trained on the proposed split.



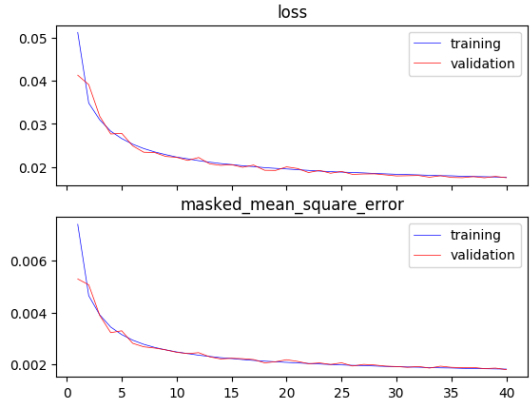
(a) Mean absolute error, $\alpha = 0$



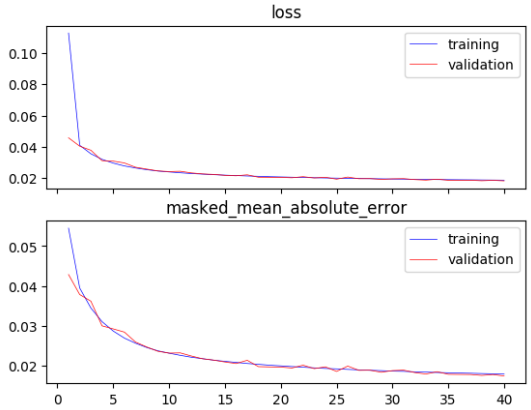
(b) Mean square error, $\alpha = 0$



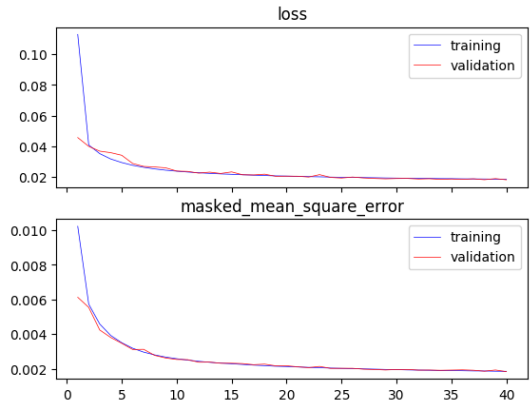
(c) Mean absolute error, $\alpha = 0.1$



(d) Mean square error, $\alpha = 0.1$

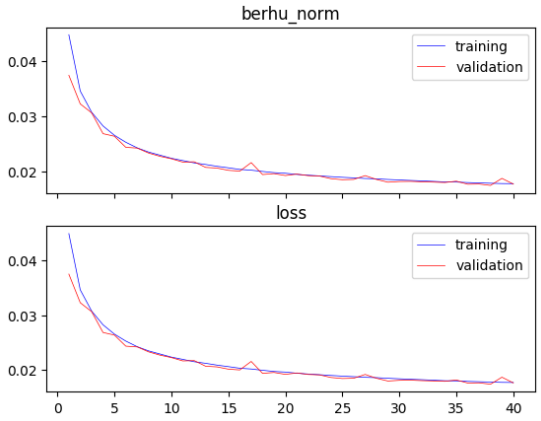


(e) Mean absolute error, $\alpha = 1$

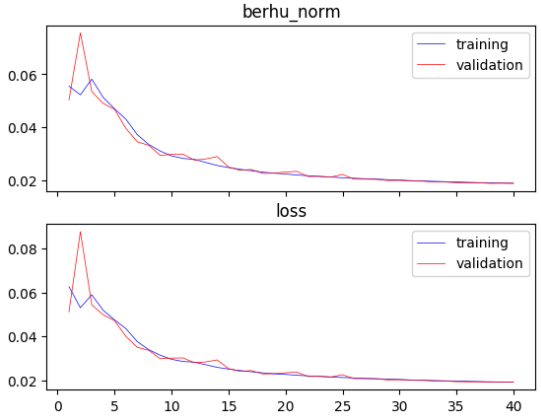


(f) Mean square error, $\alpha = 1$

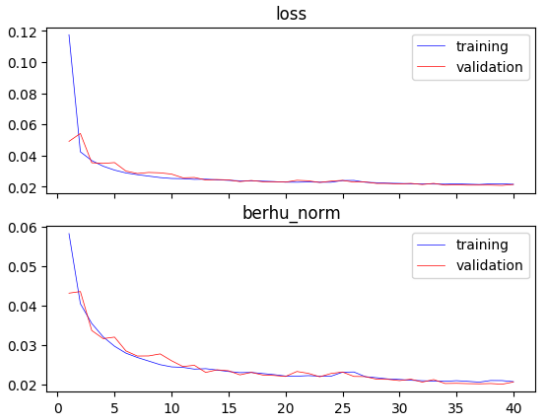
Figure 4.7: Plots for the berHu norm loss with $\alpha = [0, 0.1, 1]$ and $c = \frac{1}{5}\max_i(|y_i^* - y_i|)$ where y^* is the ground truth, y the prediction and i is the frame in the mini-batch. Network trained on the Eigen split with Godard file lists.



(a) berHu norm, $\alpha = 0$

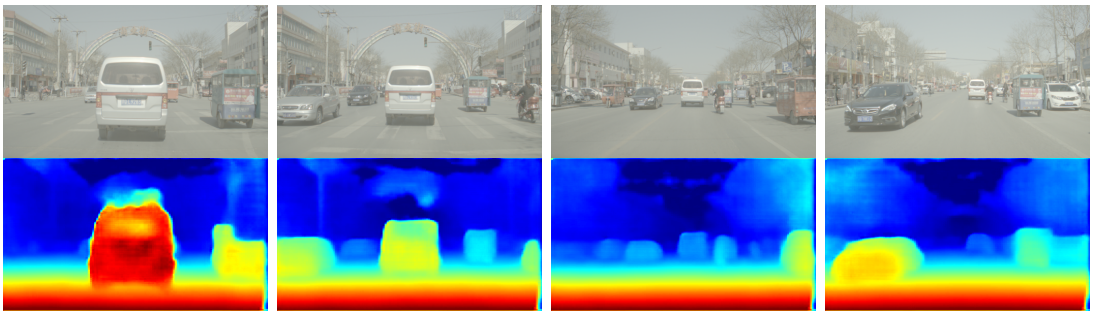


(b) berHu norm, $\alpha = 0.1$

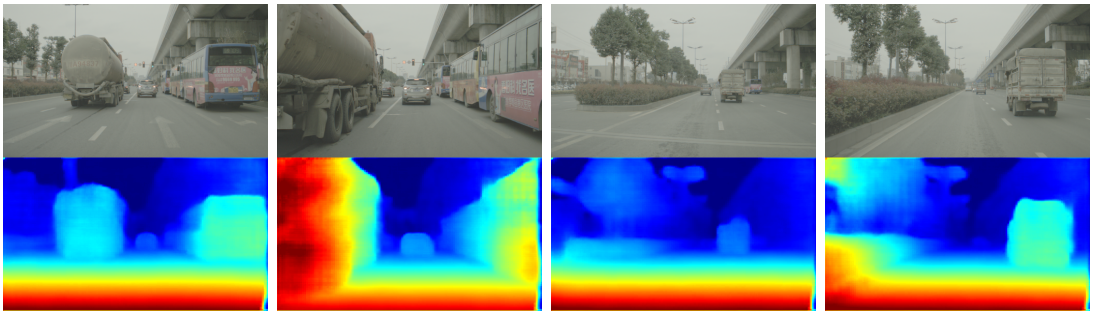


(c) berHu norm, $\alpha = 1$

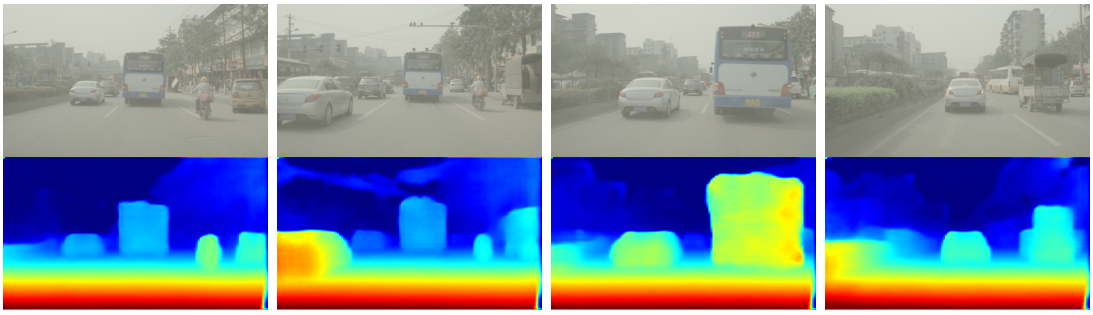
Figure 4.8: Qualitative results from the network training on the Autoliv dataset where the network succeeds to predict the depth. Images are best presented on a digital medium.



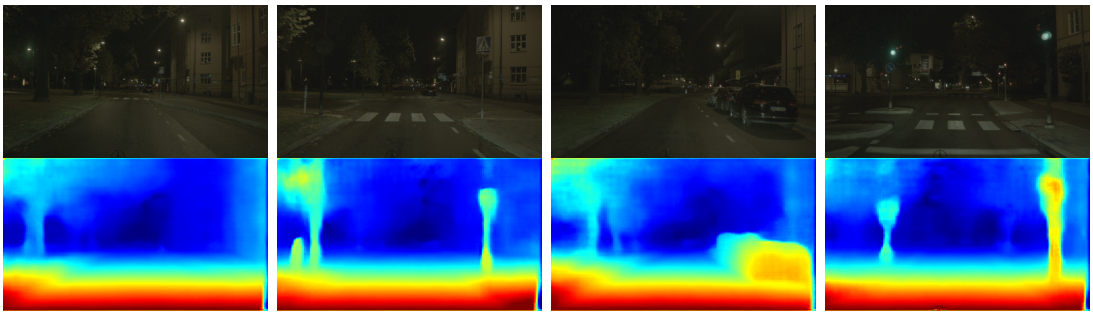
(a) (b) (c) (d)



(e) (f) (g) (h)

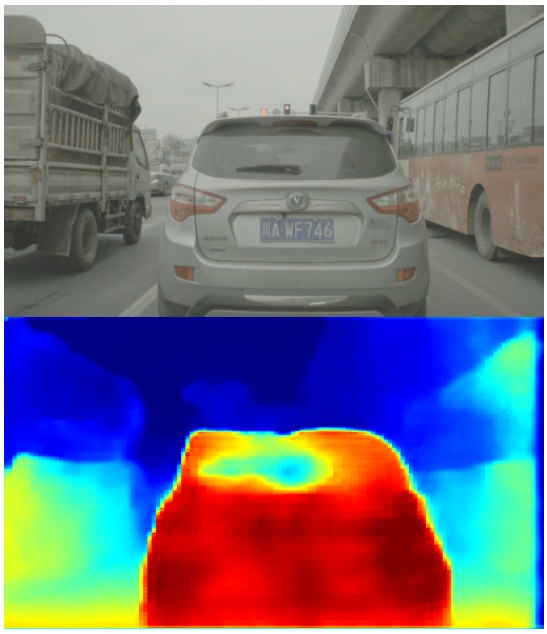


(i) (j) (k) (l)

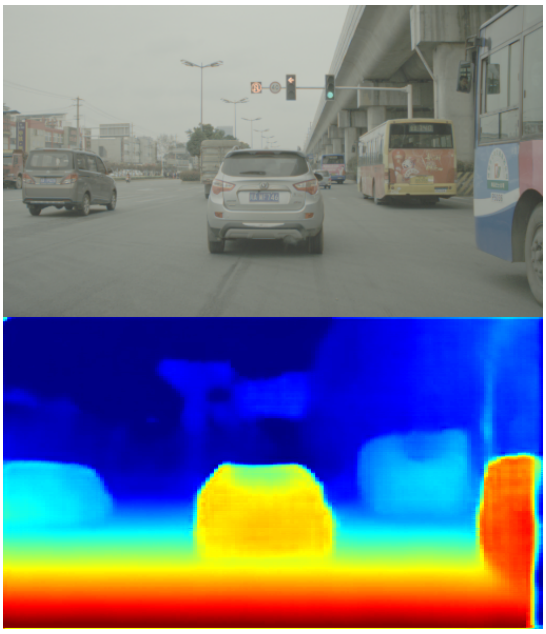


(m) (n) (o) (p)

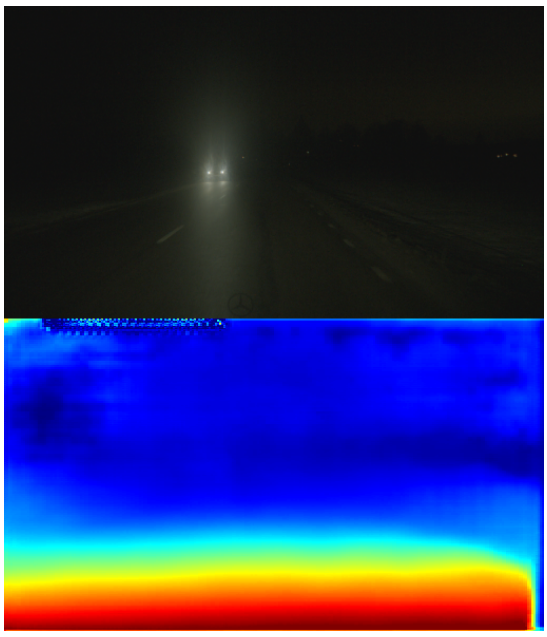
Figure 4.9: Qualitative results for the Autoliv dataset where the network mispredicted the depth. Images are best presented on a digital medium.



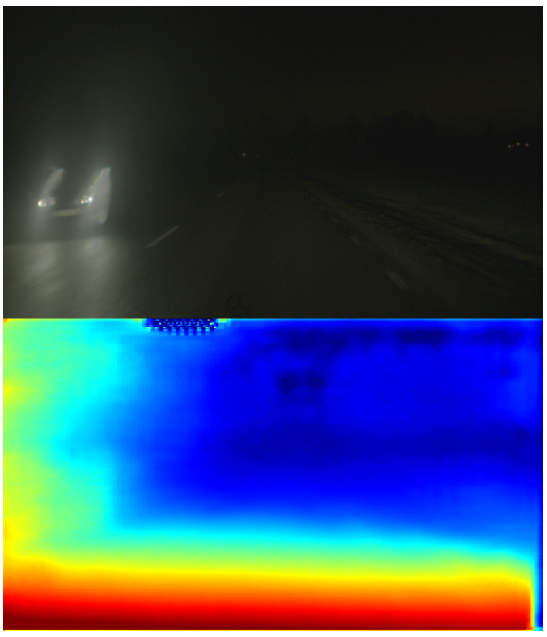
(a) The network fails to predict depth of bus and small truck.



(b) The network fails to predict depth of the upper half the bus.

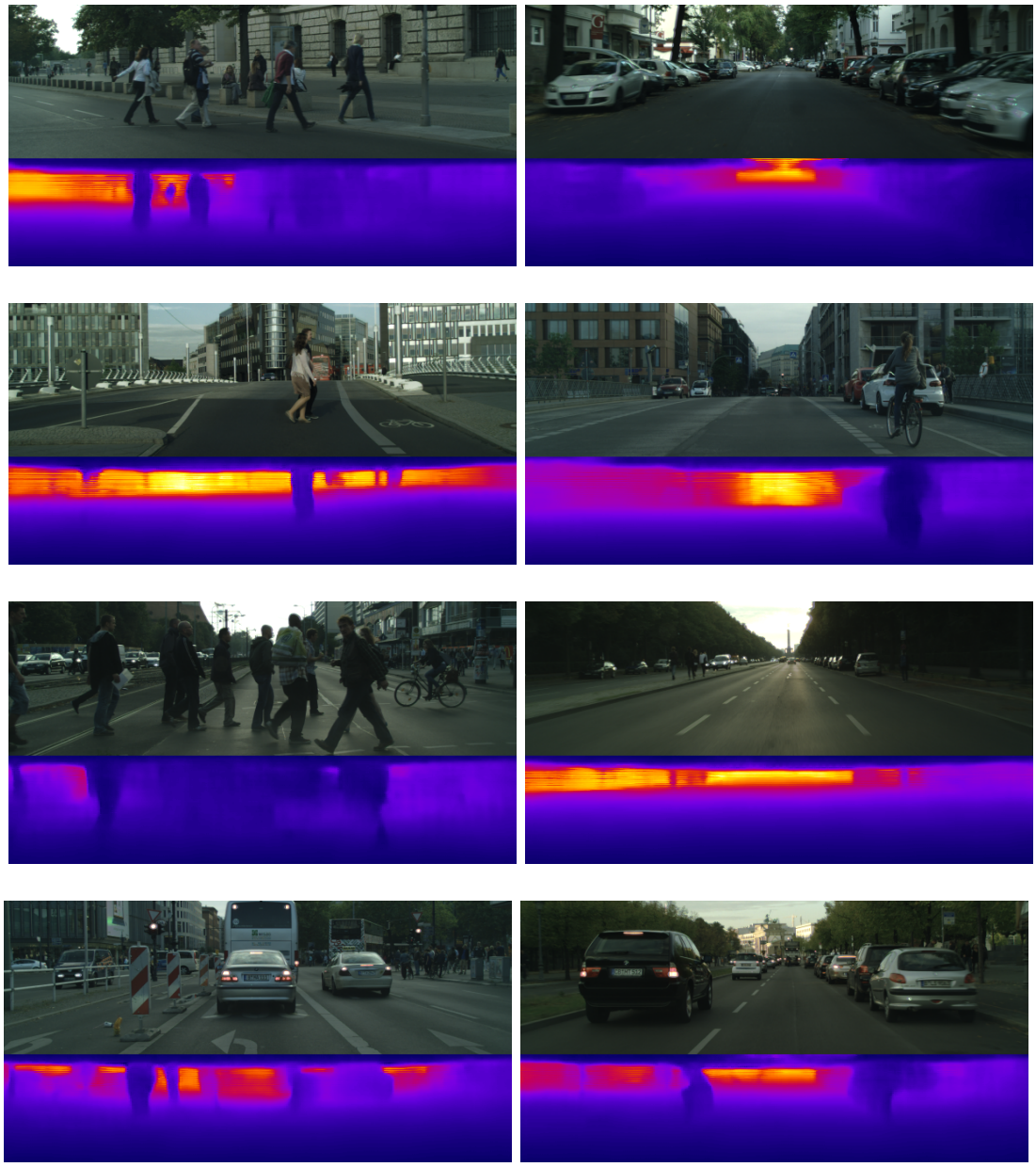


(c) The network fails to predict depth of the oncoming car.



(d) The network to predicts depth over the oncoming car.

Figure 4.10: Qualitative results of the network trained on the KITTI dataset with Eigen split, Godard file lists and 100 epochs on the Cityscapes images.



5. Discussion

The network results as well as problems encountered in the project are discussed within this section.

5.1 Depth estimation with an efficient network architecture

The general goal of the project, which was to estimate depth on single monocular images using a small and efficient neural network, has been reached. The network was able to estimate depth from single monocular images with high precision, similar to the current state-of-the-art depth estimation algorithms, both supervised and unsupervised.

Even though the high frame rate of the original implementation of ENet was not achieved in this thesis the network architecture is still very much the same. This implies that the network, with the proposed alterations, and the same implementation as Paszke *et al.* [38] should result in equivalent high frame rate performance. This shows that the network is suitable to use an embedded system such as the NVIDIA TX1 and could produce a depth estimation of a single frame in real-time.

5.2 General object detection

The network is capable of finding and estimating depth of general objects in frames. However, this is not a very robust system since the same object can be detected in one frame and be unregistered in the consecutive frame. This implies that the model is capable to estimate depth of general objects but should not solely be relied on to do so. The model could be used in conjunction with other methods such as object classification to find general objects in the road scene.

5.3 Dataset limitations

The limitations of the datasets used in the model training are discussed within this section.

5.3.1 KITTI dataset

The possible and clear limitations of the KITTI dataset are reviewed in this section.

Dataset size

The KITTI dataset was, at the time of construction, considered to be a quantitatively large dataset. The dataset is however considered fairly small for a model solving a deep learning task. The size of the dataset clearly affects the models ability to generalize, as shown in Section 4.3. This means that the KITTI dataset can be used to train a CNN that will be fairly competent in estimating depth within scenes captured in similar conditions. This is something that is not wanted since the method should be generalizable and estimate depth from all kinds of scenarios. The performance of the CNN is increased by adding more training data, as proven by [18, 27, 14] where they added the Cityscapes dataset to their unsupervised models.

LiDAR rolling shutter

The rolling shutter effect is caused by the construction of the depth recording LiDAR equipment. As described in section 3.3.1 the scanner rotates to read its surroundings. This means that it has to move over the frame to record the scene depth. The image is captured when the scanner is in the center of the frame. The angular velocity of the scanner corresponds to a scene reading time of ~ 0.1s. This is also confirmed by the difference between start and stop times in the timestamp files in the dataset. This means that a vehicle moving at 50km/h will have traveled the distance of ~ 1.4m from when the scanner started the scene recording until it has finished. The result of this is that the scanner can overlap object data points with the background data points. This effect gets more severe for higher vehicle velocities. This provides the network with a ground truth that can contain ambiguous areas where data points on the same object represent different depth values. Examples of this can be observed in Figure 5.1 where the middle image clearly shows the effect on the traffic sign that contain both object and background data points. Exactly how this effects the network training and predictions are unclear but is something that should be kept in consideration.

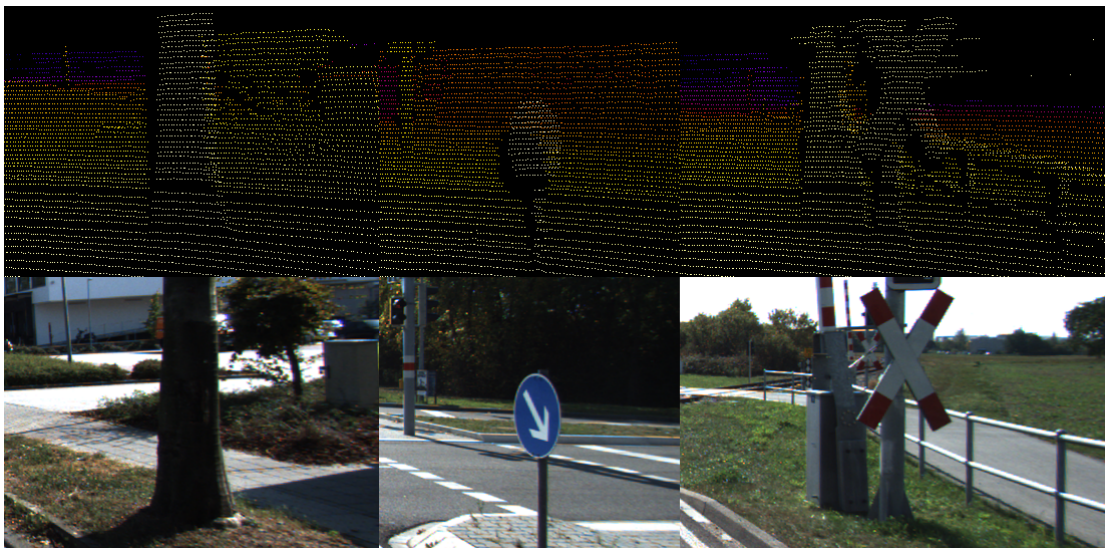


Figure 5.1: Overlapping depth data points caused by the rolling shutter effect of the LiDAR scanner when the vehicle is moving at higher speeds.

5.3.2 Autoliv dataset

The ground truth of the Autoliv dataset is a disparity depth image where the pixel intensity is inversely proportional to scene depth. The results show that the proposed network design is highly adaptable for solving the regression problem regardless of ground truth type. The accuracy of the network when training with disparity images as ground truth could trivially be improved by using a state-of-the-art disparity algorithm to generate the ground truth images. The performance could also possibly be improved by implementing the previous frame method. The Autoliv dataset has a higher frame rate than the KITTI dataset which provides a smaller difference between consecutive frames and could assist when using previous frames in network training. Adding data augmentation, such as horizontal flip, would probably increase the performance of the network since this was the case for the KITTI dataset.

However, the results show that the network has difficulties in predicting depth of rare objects and vehicles. For instance, specular reflections in large bus windows can cause the network to estimate these areas as background or buildings resulting in faulty depth estimation. The network also has problems when estimating depth for trucks with rare flatbeds (such as concrete flatbed or a tubular constructed flatbed) where no depth is predicted for the flatbed itself. Examples of the network failing to predict depth of rare objects can be observed in Figure 4.9. This is a severe problem since the knowledge of these vehicles and their distance to the observer is of crucial importance if a hazardous situation would occur. Since the network was trained on generated disparity images it also inherits the errors produced in the disparity algorithm. These errors are then amplified in the network and could be one source to the problem of disappearing rare vehicles. This problem could be partially solved by using a more accurate stereo algorithm to generate the disparity images since this does not need to be performed in real-time.

5.4 Learning limitations

The supervised and unsupervised network learning scheme provides a good scenic knowledge of the frames provided within the dataset during the training. However, when the network is introduced to frames with different scenic information it is revealed that the network has a generalization problem. Thus the network is highly dependent on the scenic input data as shown in Section 4.3 where the network performance deteriorated as the training set was reduced. This is also shown in the paper by Eigen *et al.* [13] where they use two different set of weights for analyzing indoor (NYU dataset [41]) and outdoor (KITTI dataset [15]) scenes. This shows that the network has a problem of predicting depth for objects or scenes it has not been trained on.

The network was evaluated on the Cityscapes dataset [8] in order to evaluate the generalization capability. The network was able to estimate depth on the Cityscapes images, despite the images being captured on a camera with different focal length and mounting position. The qualitative performance of some of the Cityscapes images are previewed in Figure 4.10 and show cases when the network makes a fairly accurate prediction (qualitatively) and where it does not. The network is in other words sometimes capable and sometimes not capable in predicting depth for images captured by a camera it has not been trained on. The network has a fairly

weak generalization capability but is not completely fruitless.

5.5 Input data variation studies

The results from the ablation studies are reviewed in this section.

5.5.1 Right frame

The network trained using the right image in the training set produced predictions that were not as accurate when compared to the regular network training. This could be explained by the stereo camera setup used in the KITTI dataset. The stereo rig had a base line, that is the distance between the left and right camera, of 54 cm. This base line provided a very large disparity between the image pair. The LiDAR scanner alignment provided a distance of 60 cm from the scanner to the right color camera causing a considerable disparity between the ground truth and the right frame. The reduced performance was qualitatively evident since the predictions lacked horizontal sharp edges around cars and other objects. This effect would likely be caused by the large disparity between the left and right frame since the right camera was horizontally displaced from the left camera. The qualitative results are presented in Section 4.4.2.

5.5.2 Previous frame

Adding previous frames in the network training was made to hopefully provide structure from motion information to the network. However, this did not succeed well on the KITTI dataset. The results indicate that the predictions generated by the network trained on the KITTI data set with a 10 fps rate is negatively affected by the use of a previous frame. Reasons for this could be that the scenic difference between following frames in the dataset is large. As discussed in Section 5.3.1, the distance traveled by the vehicle at higher speeds is considerable and clearly affects the scenic difference between consecutive frames. Hence the network is presented with two considerably different scenes with the same ground truth which would probably not contribute to the network training process. The resulting predictions from the network trained using one previous frame has quantitative and qualitative results that are worse when compared to the network trained without previous frames. The quantitative results can be observed in Section 4.4.1 and the qualitative results in Figure 4.5g, 4.5h and 4.5i.

5.5.3 Reduced training set

It can be observed in Table 4.3 (and Figure 4.4) that the training performance increase as the number of training samples increased. This indicates that more data provides a better model generalization and argue for the importance of a large dataset in order to create a generalized model for depth estimation. The network training on the Ω_{16140} subset performed better than the training on the entire dataset. The reason for this is however unclear.

5.5.4 Parameter search

The parameter search performed indicated which parameter combination resulted in the best network training settings. Other order of magnitudes of λ could have been evaluated in a more extensive parameter search but was not carried out due to the time complexity of such a search. Single network trainings were evaluated with $\lambda = 10$. This regularization however proved to be too intense and resulted in a smeared prediction with poor visual performance and was not investigated further.

5.6 Prior accuracy evaluation work

The previous work done on the subject of depth estimation using CNN's [13, 12, 27, 18, 29, 14] have primarily only evaluated the network performance using Equations 3.3. Most of the papers have failed to show where the network performs well and only show the quantitative results along with some qualitative results. Godard *et al.* [18] visualize the prediction uncertainty of their model but do not show the prediction error compared to the ground truth. This could be misleading since the most important area is in front and the close vicinity of the vehicle and should therefore be prioritized. A quantitative performance analysis of the entire frame without spatial locality, or prioritized areas, taken into consideration can produce misleading conclusions. As an example there are some scenes in the KITTI dataset that contain large fields. These fields are of little importance to the depth estimation problem but generate large errors that adversely affect the quantitative results of the network. The proposed accuracy thresholds in previous papers are also too low to be relevant in real depth estimation scenarios. An accuracy error of $\pm 25\%$, that is the previous lowest threshold value, could result in an error of several meters. This could lead to severe consequences in an auto emergency braking (AEB) situation.

Possible solutions to this problem is to implement higher precision thresholds, proposed in Section 3.6

6. Conclusions

The proposed network architecture proved to be a suitable method for depth estimation of single monocular images. The network was able to predict depth with a close to state-of-the-art accuracy while only utilizing a fraction of the learnable parameters compared to the other CNN solutions. The lightweight network architecture is highly usable on low powered systems such as an Internet of Things (IoT) device or an embedded system. The network was able to detect general objects but not in a sufficiently robust manner to solely be used for general objects detection and depth estimation. The implemented architecture is also very small and computationally lightweight as proven by Paszke *et al.* [38].

6.1 Future work

Future improvements to the proposed model are reviewed in this section.

Previous frame

Using a previous frame could perhaps be more efficient for sequences with a higher frame rate since the differences between consecutive frames would be less significant.

Parameter search

A finer parameter search of the regularization parameter λ could be performed resulting in a more optimized network. This was not performed simply because of the time such a parameter search would consume on the equipment used for network training.

Pose as a classification problem

The depth estimation problem could also be solved as a classification problem. One downside is that a discretization of the possible distance measurements would be performed. This would result in many classes for the pixel-wise classification task, possibly increasing the problem complexity. The upside would be a confidence value added to each predicted pixel that would reveal the network confidence in the prediction. This is absent when solving the regression problem.

Spatial dropout removal

The removal of spatial dropout in the network architecture could be evaluated together with the $\mathcal{L}_2$ norm. This would be interesting to investigate since the $\mathcal{L}_2$ norm is sensitive to regular dropout [24].

Data augmentation

Further data augmentation could be added to the network training. The only data augmentation used in the network training was the horizontal flip with probability of 50%. Other types of data augmentations could be implemented. Left-right tilt, gamma, brightness and color augmentations could increase the network performance due to the increase of available training data. It is also of consideration to add intensity and spatial scaling augmentation to increase the number of training frames used in the network training even further.

A. Appendix

Data splits

The two data splits used in the report. The file lists to the KITTI and Eigen splits are provided by [18]. The proposed balanced split is presented in Table A.1.

Table A.1: The data split for data generalization. Sequences marked in bold were used in the validation set.

Training set	Test set
2011_09_26_drive_0001	2011_09_26_drive_0027
2011_09_26_drive_0002	2011_09_26_drive_0028
2011_09_26_drive_0005	2011_09_26_drive_0029
2011_09_26_drive_0009	2011_09_26_drive_0035
2011_09_26_drive_0011	2011_09_26_drive_0036
2011_09_26_drive_0013	2011_09_26_drive_0039
2011_09_26_drive_0014	2011_09_26_drive_0046
2011_09_26_drive_0015	2011_09_26_drive_0079
2011_09_26_drive_0019	2011_09_26_drive_0084
2011_09_26_drive_0020	2011_09_26_drive_0091
2011_09_26_drive_0022	2011_09_26_drive_0093
2011_09_26_drive_0023	2011_09_26_drive_0095
2011_09_26_drive_0032	2011_09_26_drive_0096
2011_09_26_drive_0048	2011_09_26_drive_0104
2011_09_26_drive_0051	2011_09_26_drive_0106
2011_09_26_drive_0052	2011_09_26_drive_0113
2011_09_26_drive_0056	2011_09_26_drive_0117
2011_09_26_drive_0059	2011_09_28_drive_0001
2011_09_26_drive_0061	2011_09_28_drive_0002
2011_09_26_drive_0064	2011_09_29_drive_0004
2011_09_26_drive_0070	2011_09_29_drive_0071
2011_09_26_drive_0086	2011_09_30_drive_0016
2011_09_26_drive_0087	2011_09_30_drive_0018
2011_09_26_drive_0101	2011_09_30_drive_0020
2011_09_30_drive_0034	2011_09_30_drive_0027
2011_10_03_drive_0027	2011_09_30_drive_0028
2011_10_03_drive_0034	2011_09_30_drive_0033
2011_10_03_drive_0042	2011_10_03_drive_0047

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *Tensorflow: Large-scale machine learning on heterogeneous distributed systems*, 2015.

[2] Autoliv, *Active Safety Systems - Vision Systems*, 2017.

[3] V. Badrinarayanan, A. Kendall, and R. Cipolla, *Segnet: A deep convolutional encoder-decoder architecture for image segmentation*, arXiv preprint arXiv:1511.00561, (2015).

[4] A. Canziani, A. Paszke, and E. Culurciello, *An analysis of deep neural network models for practical applications*, arXiv preprint arXiv:1605.07678, (2016).

[5] Y. Cao, Z. Wu, and C. Shen, *Estimating depth from monocular images as classification using deep fully convolutional residual networks*, arXiv preprint arXiv:1605.02305, (2016).

[6] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, *Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs*, arXiv preprint arXiv:1606.00915, (2016).

[7] F. Chollet, *Keras*, 2015.

[8] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, *The cityscapes dataset for semantic urban scene understanding*, in Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[9] Creative Commons Corporation, *CC0 1.0 Universal*.

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, *Imagenet: A large-scale hierarchical image database*, in Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on, IEEE, 2009, pp. 248–255.

[11] V. Dumoulin and F. Visin, *A guide to convolution arithmetic for deep learning*, arXiv preprint arXiv:1603.07285, (2016).

[12] D. Eigen and R. Fergus, *Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture*, in Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 2650–2658.

[13] D. Eigen, C. Puhrsch, and R. Fergus, *Depth map prediction from a single image using a multi-scale deep network*, in Advances in neural information processing systems, 2014, pp. 2366–2374.

[14] R. Garg, G. Carneiro, and I. Reid, *Unsupervised cnn for single view depth estimation: Geometry to the rescue*, in European Conference on Computer Vision, Springer, 2016, pp. 740–756.

[15] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, *Vision meets robotics: The kitti dataset*, The International Journal of Robotics Research, 32 (2013), pp. 1231–1237.

[16] A. Geiger, P. Lenz, and R. Urtasun, *Are we ready for autonomous driving? the kitti vision benchmark suite*, in Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on, IEEE, 2012, pp. 3354–3361.

[17] X. Glorot, A. Bordes, and Y. Bengio, *Deep sparse rectifier neural networks*, in Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, 2011, pp. 315–323.

[18] C. Godard, O. Mac Aodha, and G. J. Brostow, *Unsupervised Monocular Depth Estimation with Left-Right Consistency*, 2016.

[19] R. H. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung, *Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit*, Nature, 405 (2000), p. 947.

[20] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition. corr abs/1512.03385 (2015)*, 2015.

[21] ——, *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*, in Proceedings of the IEEE international conference on computer vision, 2015, pp. 1026–1034.

[22] I. P. Howard, *Perceiving in depth, volume 3: Other mechanisms of depth perception*, Oxford University Press, 2012.

[23] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, arXiv preprint arXiv:1502.03167, (2015).

[24] A. Karpathy, *Stanford corse cs231n convolutional neural networks for visual recognition*, January 2017.

[25] D. Kingma and J. Ba, *Adam: A method for stochastic optimization*, arXiv preprint arXiv:1412.6980, (2014).

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, in Advances in neural information processing systems, 2012, pp. 1097–1105.

[27] Y. Kuznietsov, J. Stückler, and B. Leibe, *Semi-supervised deep learning for monocular depth map prediction*, arXiv preprint arXiv:1702.02706, (2017).

[28] L. Ladicky, J. Shi, and M. Pollefeys, *Pulling things out of perspective*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 89–96.

[29] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab, *Deeper depth prediction with fully convolutional residual networks*, in 3D Vision (3DV), 2016 Fourth International Conference on, IEEE, 2016, pp. 239–248.

[30] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning*, nature, 521 (2015), p. 436.

[31] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE, 86 (1998), pp. 2278–2324.

[32] B. Li, C. Shen, Y. Dai, A. van den Hengel, and M. He, *Depth and surface normal estimation from monocular images using regression on deep features and hierarchical crfs*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1119–1127.

[33] F. Liu, C. Shen, and G. Lin, *Deep convolutional neural fields for depth estimation from a single image*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 5162–5170.

[34] F. Liu, C. Shen, G. Lin, and I. Reid, *Learning depth from single monocular images using deep convolutional neural fields*, IEEE transactions on pattern analysis and machine intelligence, 38 (2016), pp. 2024–2039.

[35] J. Long, E. Shelhamer, and T. Darrell, *Fully convolutional networks for semantic segmentation*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 3431–3440.

[36] A. L. Maas, A. Y. Hannun, and A. Y. Ng, *Rectifier nonlinearities improve neural network acoustic models*, in Proc. ICML, vol. 30, 2013.

[37] S. A. Papert, *The summer vision project*, (1966).

[38] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, *Enet: A deep neural network architecture for real-time semantic segmentation*, arXiv preprint arXiv:1606.02147, (2016).

[39] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, *ImageNet Large Scale Visual Recognition Challenge*, International Journal of Computer Vision (IJCV), 115 (2015), pp. 211–252.

[40] A. Saxena, M. Sun, and A. Y. Ng, *Make3d: Learning 3d scene structure from a single still image*, IEEE transactions on pattern analysis and machine intelligence, 31 (2009), pp. 824–840.

[41] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, *Indoor segmentation and support inference from rgbd images*, in European Conference on Computer Vision, Springer, 2012, pp. 746–760.

[42] S. Singh, *Critical reasons for crashes investigated in the national motor vehicle crash causation survey*, tech. rep., 2015.

[43] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, *Rethinking the inception architecture for computer vision*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2818–2826.

[44] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler, *Efficient object localization using convolutional networks*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 648–656.

[45] Velodyne Acoustics, Inc., *HDL-64E User's manual*, 345 Digital Drive, Morgan Hill, CA 95037.

[46] P. Wang, X. Shen, Z. Lin, S. Cohen, B. Price, and A. L. Yuille, *Towards unified depth and semantic prediction from a single image*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 2800–2809.

[47] F. Yu and V. Koltun, *Multi-scale context aggregation by dilated convolutions*, arXiv preprint arXiv:1511.07122, (2015).

[48] M. D. Zeiler and R. Fergus, *Visualizing and understanding convolutional networks*, in European conference on computer vision, Springer, 2014, pp. 818–833.

[49] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, *Unsupervised learning of depth and ego-motion from video*, arXiv preprint arXiv:1704.07813, (2017).