CHALMERS
UNIVERSITY OF TECHNOLOGY

Self-Supervised Cross-Connected CNNs for Binocular Disparity Estimation

Master’s thesis in Systems, Control and Mechatronics

TRYGVE GRÖNDAHL
ANNA SAMUELSSON

Department of Mechanics and Maritime Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

Master's thesis 2018:26

Self-Supervised Cross-Connected CNNs for Binocular Disparity Estimation

TRYGVE GRÖNDAHL

ANNA SAMUELSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Adaptive Systems Research Group
Chalmers University of Technology
Gothenburg, Sweden 2018

Self-Supervised Cross-Connected CNNs for Binocular Disparity Estimation
TRYGVE GRÖNDAHL
ANNA SAMUELSSON

Supervisor: Oskar Noresson, Yaowen Xu, CPAC Systems AB
Examiner: Peter Forsberg, Adaptive Systems

Master's Thesis 2018:26
Department of of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Adaptive Systems Research Group
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The left image from an image pair in the Cityscapes dataset. A depth for every pixel was calculated using a self-supervised X-CNN.

Typeset in LaTeX
Gothenburg, Sweden 2018

Self-Supervised Cross-Connected CNNs for Binocular Disparity Estimation
TRYGVE GRÖNDAHL
ANNA SAMUELSSON
Department of Mechanics and Maritime Sciences
Chalmers University of Technology

Abstract

When developing autonomous vehicles, sensors with high accuracy and speed are needed. One type of sensor that can gather a lot of information is the camera. From two stereo images a disparity between them can be calculated, and from that the depth. The drawback with today's algorithms is the trade-off between high quality estimation and computational speed.

By taking inspiration from recently published neural networks for other applications, we present a novel design for disparity estimation networks. We design a cross-connected convolutional neural network to calculate full HD disparity maps from stereo images at a high frequency. By transfer training the network, using self-supervised learning, the network can learn to handle new environments. The network shows significantly faster runtimes than other disparity estimation networks, with the loss of some accuracy.

We show that the self-supervised loss functions perform poorly when the images are not aligned, which is important to solve for real life applications of the network. Furthermore, we present ideas on how to improve the network's runtime further.

Keywords: Artificial Neural Network (ANN), Convolutional Neural Network (CNN), deep learning, self-supervised, machine learning, cross-connection, stereo vision, disparity estimation

Acknowledgements

We would not have achieved this thesis alone, and would like to thank those who made this thesis possible. First, we would like to thank our supervisors Oskar Noresson and Yaowen Xu for all help and support during our work. We would also like to thank our examiner Peter Forsberg who has been very engaged in our project and our way forward.

This thesis work was conducted at the company CPAC Systems AB, where we have been well received and given a lot of support and interest, we therefore thank the company as such for making this thesis possible and for making us feel welcome.

Trygve Gröndahl and Anna Samuelsson, Gothenburg, May 2018

Thesis advisors: Oskar Noresson, Yaowen Xu, CPAC Systems AB
Thesis examiner: Peter Forsberg, Adaptive Systems

Abbreviations

ANN	Artificial Neural Network
CNN	Convolutional Neural Network
EPE	Endpoint error
FOV	Field Of View
FT3D	FlyingThings3D
Gbg	Gothenburg
GC-net	Geometry and Context network
px	pixels
ReLU	Rectified Linear Unit
RGB	Red Green Blue, used for color images
SIFT	Scale-Invariant Feature Transform
X-CNN	Cross Convolutional Neural Network

Contents

List of Figures

List of Tables

1 Introduction

During recent years, many companies in the automotive industry have invested heavily in the development of autonomous vehicles. Volvo Group, for example, has started projects in developing autonomous trucks for usage in mining [1] and in garbage collection [2]. An autonomous system could improve the traffic safety and replace human drivers in hazardous or inhospitable environments.

The system of an autonomous vehicle is safety critical, meaning that a failure of such a system could lead to irreversible damage [3]. Safe systems needs to be both redundant and robust, which can be achieved by combining the result from different sensors and algorithms. Different types of sensors have different strengths and weaknesses, hence the usage of different sensors is needed to create a robust digital representation of the world around the system. Having multiple sensors to cover different cases is important, as the data they provide is useful in different situations. An example is the sensors an autonomous car could use. In a tunnel a GPS will not be of much use, as the satellite signals will not reach it, whereas the laser of a LiDAR will be reflected on the tunnel walls and provide useful data. In an open landscape the reverse would be true. GPS and LiDAR are just two examples of useful sensors in this application, others can be cameras, radar and IR-sensors. One of the strengths of using cameras as sensors is the amount of data that can be obtained from images in a variety of environments. Drawbacks exist however, such as the dependency of sufficient light.

Cameras can be used for many applications and multiple cameras can be combined to aggregate data. The use of two or more sources for vision, such as cameras, is known as stereo vision. By using the difference of where an object is located in the images from both sources, a position of the object relative to the cameras can be calculated. By changing the distance between the sources of vision, the quality of the estimation at different depths change. Cameras further apart gives better depth estimation of objects further away. By using multiple stereo image pairs at different distance from each other, the depth estimation can be refined and handle a wider range of distances [4].

The introduction of convolutional neural networks (CNNs) has made cameras and computer vision a strong tool in creating an accurate digital interpretation of the world [5]. When training a neural network, such as a CNN, the network is fed with large quantities of data and can learn features in or properties of the data. In order to train a network to calculate the depth from images, different learning strategies can be employed depending on the available datasets. Recent research in supervised stereo vision shows very high precision with a sub-pixel accuracy for the depth estimation [6], but at the cost of being very computationally heavy. The estimations can also be done using unsupervised training which has shown interesting results for both stereo and monocular vision [7, 8].

Problems with calculating depth using computer vision still exist. Cameras as sensors have various drawbacks, especially in varying light and weather conditions. The conventional algorithms for calculating depth are often computationally heavy, and insufficient for images with occluded or texture-less areas. The use of CNNs might tackle these problems if trained with a sufficient amount of data, yielding the problem of finding diverse, publicly available real-life data with ground truth depth. Training a network with only synthetic data will not generalize to real life [9], and one way of solving the problem of finding ground truth data could be to use unsupervised learning.

1.1 Purpose

The purpose of this project is to examine how neural networks can be used to obtain depth estimations from a stereo camera pair and how it can be implemented on an embedded platform. This will be done by implementing existing neural network theory and research followed by examining the possibilities of refining the accuracy and runtime of the estimation. The aim is to use unsupervised learning to create accurate depth maps in new environments, thus eliminating the need for labeled datasets.

1.1.1 Goals

The main goal of the project is to design and implement a neural network for depth estimation using high resolution images in an embedded system. The embedded approach leads to restrictions on the computational power and the runtime of the system. The goals are:

- Predict a full HD, 1920×1080 pixels, disparity map
- Predict with a frequency of 10 Hz, i.e. generate 10 disparity maps every second, on a NVIDIA Jetson TX2

1.2 Scope

The project will focus on computer vision using cameras as the primary sensors. The scope of this Master's thesis does not involve the design of the hardware used, as this will be provided. Thus, the camera specifications, distance between cameras and other hardware design parameters are set.

1.3 Thesis outline

This thesis consists of five parts. The introductory part presents a brief background and the purpose and scope of the thesis. In the theory chapter, binocular stereo vision and artificial neural networks is presented. These two areas are then combined to a section about the theory behind stereo vision using neural networks, emphasizing convolutional neural networks. The method chapter describes the work flow of the thesis and the methods used to obtain the result and the strategic decisions made. In the method method the different network designs are presented as well as some initial results and evaluations of those designs. The results are then presented, compared and discussed in the fourth chapter. The last chapter contains a conclusion of the work done and a presentation of ideas about future work.

2
Theory

In this chapter the theoretical background of the project is presented. There are three main topics in this chapter. The first section presents the camera hardware, calibration of the cameras and the use of stereo vision for depth estimation. In the second section the theory of artificial neural networks is described emphasizing on convolutional neural networks and methods of training networks. The last section describes the use of convolutional neural networks for disparity estimation together with recent research in the area.

2.1 Stereo vision

In this section a simple yet efficient camera model is presented and explained together with the drawbacks of that model and how to compensate for those. Furthermore, the idea behind camera calibration, both for one camera and for two cameras in stereo, is described. Finally, binocular disparity and how it is calculated is presented.

2.1.1 A simple camera model

A simple camera model often used is that of a pinhole camera, which is a model of a box with a tiny aperture in one of its walls [10]. No light can pass through the wall except for through the hole. The light passing through the hole is projected on the opposite wall of the box, called the image plane. In reality, a camera needs more light than available in this simplified model. By using lenses, cameras can acquire the amount of light needed. The lens introduces distortions in the resulting image, which it is preferable to compensate for [11].

2.1.1.1 The pinhole model

In Figure 2.1 a representation of the pinhole and inverted pinhole camera models can be found. The light from outside the box is passed through the aperture and hits the imaging plane a distance f, the focal length, from the aperture. The pinhole camera will produce an image of the scenery upside down. The idea behind the inverse pinhole camera is to project the image plane to the other side of the pinhole, yielding the inverse of the image. This is the model often used for calculations, since the inverse model makes the calculations easier [10]. The field of view (FOV) is the imaging limit for an imaging system, depending on the lenses or sensors used. The

FOV and the focal length, f is related to each other. For the horizontal FOV the equation can be written as

$$FOV_{\text{horizontal}} = 2\arctan\left(\frac{\text{width}}{2f}\right), \tag{2.1}$$

where *width* is the width of the resulting image in millimeters [12].

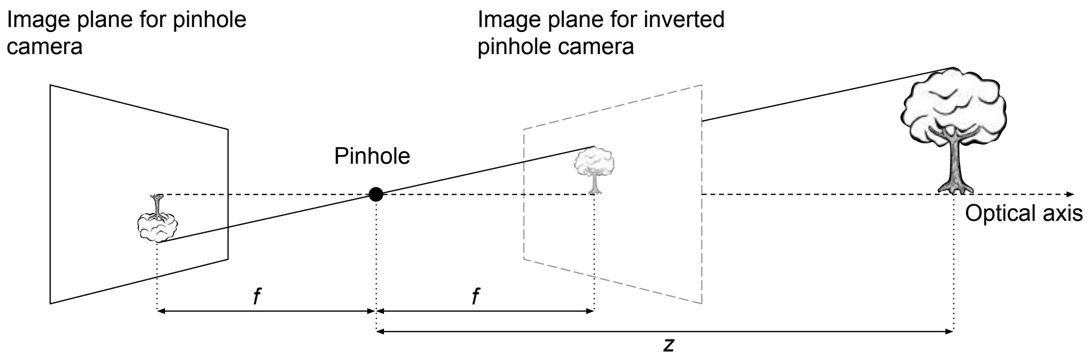


Image plane for pinhole camera
Image plane for inverted pinhole camera
Pinhole
Optical axis
f
f
z

Figure 2.1: A visualization of how a pinhole camera model works. The black dot is the pinhole, and the image plane is the focal length f from it. In the image plane of the pinhole camera the tree will be upside down. However, this can be solved by using the inverse pinhole camera model, where the image plane is moved in front of the pinhole.

In reality the focal lengths in the x and y directions are different, f_x and f_y, since the shape of individual pixels in a camera often are non-square. The variables c_x and c_y are used for describing the possible offset in the middle point in the image plane compared to the optical axis, called principal point. The coordinates of a real world point P that has the position (X, Y, Z) can be used to calculate the coordinates in the image, x and y, as

$$\begin{aligned} x &= f_x \frac{X}{Z} + c_x \\ y &= f_y \frac{Y}{Z} + c_y, \end{aligned} \tag{2.2}$$

using the inverse pinhole camera model with the origin in the aperture. These equations can be rewritten into a matrix that can be used for translating between a real world point and the corresponding point in the image plane. The matrix is often denoted $\boldsymbol{M}$ and consists of the *intrinsic parameters*; f_x, f_y, c_x and c_y [10]. The transformation is written as

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}. \tag{2.3}$$

In most applications it is preferred to transform these camera coordinates, to coordinates in the world coordinate system. This can be done by linear transformation using a rotation matrix $\boldsymbol{R}$ and a translation vector $\boldsymbol{t}$, called the *extrinsic parameters*. The transformation from world coordinates to camera coordinates can be written as $\boldsymbol{P}_{\text{camera}} = \boldsymbol{R}(\boldsymbol{P}_{\text{world}} - \boldsymbol{t})$. $\boldsymbol{R}$ and $\boldsymbol{t}$ are calculated for the specific camera through calibration [13].

2.1.1.2 Lens distortion

There are two types of common lens distortions [11]. Radial distortion is the most common and has the highest effect at the edges of the images, leading to the "barrel" or "fish-eye" effect. Most lenses bend light more the further away from the center the light hits the lens which causes this distortion. The formula for compensating for this is a Taylor series expansion in the neighborhood of r=0;

$$\begin{aligned} x_{\text{corrected}} &= x(1 + k_1 r^2 + k_2 r^4 + k_3 rv), \\ y_{\text{corrected}} &= y(1 + k_1 r^2 + k_2 r^4 + k_3 rv), \end{aligned} \tag{2.4}$$

where k_3 is commonly neglected for cameras without a special fish-eye lens [11].

The second most common lens distortion is tangential distortion originating from the lens not being exactly parallel to the imaging plane. The two equations for correction of this distortion are:

$$\begin{aligned} x_{corrected} &= x + 2p_1 y + p_2(r^2 + 2x^2, \\ y_{corrected} &= y + p_1(r^2 + 2y^2) + 2p_2 x. \end{aligned} \tag{2.5}$$

The parameters k_1, k_2, k_3, p_1 and p_2 are also called intrinsic and are calculated through the camera calibration [13]. Other kinds of lens distortions exists but do not usually effect the resulting image enough to be compensating for.

2.1.2 Camera calibration

When capturing images to create a dataset, it is important that the cameras are calibrated. The calibration of a camera is done to find the intrinsic and extrinsic parameters for that specific camera [13]. In this section the main idea of camera calibration will be explained, but not the mathematical theory behind all steps. Often, a library such as OpenCV [14] is used for calibration. The calibration procedure consists of taking a set of photos of an object with known dimensions, often in the form of a 2D "chessboard" with alternating white and black squares. When translating and rotating the object, the fact that the dimensions of the object is known can be used to estimate the unknown intrinsic parameters [15].

Stereo calibration is the process of finding a relationship between two cameras in space. The stereo calibration is often followed by rectification, aligning the image planes of the cameras. The calibration is done by calculating the rotation matrix and translation vector between two cameras, calculated as:

$$\boldsymbol{R} = \boldsymbol{R_r} \boldsymbol{R_l}^T \tag{2.6}$$

$$\boldsymbol{t} = \boldsymbol{t_r} - \boldsymbol{R}\boldsymbol{t_l}, \tag{2.7}$$

where $\boldsymbol{R}$ and $\boldsymbol{t}$ denotes the rotation and translation that brings the right camera coordinate system into the left one [16]. $\boldsymbol{R_r}$, $\boldsymbol{t_r}$, $\boldsymbol{R_l}$ and $\boldsymbol{t_l}$ are the rotation matrices and translation vectors found out by two single camera calibrations, used to transform a point's world coordinates to camera coordinates. These are used as:

$$\boldsymbol{P}_{\text{left\_camera}} = \boldsymbol{R_l}(\boldsymbol{P}_{\text{world}} - \boldsymbol{t_l}) \tag{2.8}$$

$$\boldsymbol{P}_{\text{right\_camera}} = \boldsymbol{R_r}(\boldsymbol{P}_{\text{world}} - \boldsymbol{t_r}). \tag{2.9}$$

2.1.3 Disparity and depth calculation

Two cameras that are aligned in stereo and that has been calibrated, can be used together in order to produce a depth map. The images the cameras produce can be combined by finding the corresponding pixels in the images [17]. Distinctive patterns such as edges or points, known as features, in the images are often used for matching. A number of different matching algorithms exist, such as the well known scale-invariant feature transform (SIFT) [18]. The distance between the corresponding pixels in two images are called the binocular disparity. In Figure 2.2 two cameras registering a point P can be seen from a top-view. The camera to the left is called C_l and the one to the right C_r. The cameras are placed a distance b from each other, known as the baseline. The focal length f is the same for both cameras. The difference between a point in the left image p_l, having pixel coordinates (x_l, y_l), and the same point in the right image p_r, having pixel coordinates (x_r, y_r), is the disparity, denoted d [19]. Assuming that the two cameras have been rectified, $y_l = y_r$ the disparity for p_l i defined as:

$$d(p_l) = x_l - x_r. \tag{2.10}$$

Note that the binocular disparity is only measured in horizontal direction. The

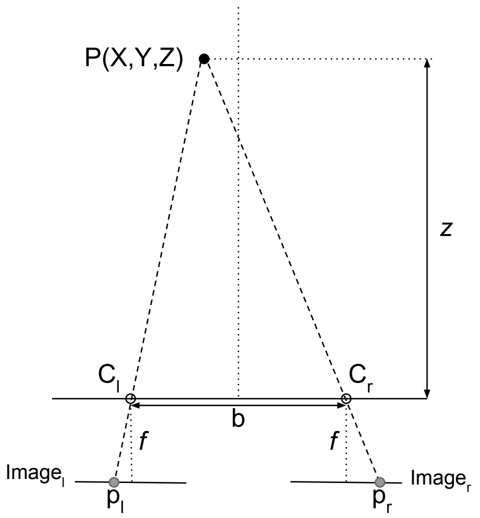


Figure 2.2: A top-view of two cameras photographing a point P a distance z from the camera baseline. The cameras produce an image each and the point P will have different coordinates in those images, since it has been seen from different views.

distance z from the camera baseline to the point P can be calculated as

$$z = \frac{bf}{p_l - p_r} = \frac{bf}{d(p_l)}[20]. \tag{2.11}$$

The stereo disparity estimation algorithms are commonly divided into four building blocks. Most algorithms focus on a subset of these [21]:

- Matching cost computation
- Cost aggregation
- Disparity computation or optimization
- Disparity refinement

This way of dividing the disparity estimation algorithm is often used both for neural networks and in conventional image analysis methods.

2.2 Artificial neural networks

Artificial neural networks (ANN) is one part in the area of machine learning. The idea behind ANNs is to imitate the neurons and synapses of the human brain [22]. ANNs have a graph-like structure with layers of neurons connected with different weights. An example of how an ANN can be structured is shown in Figure 2.3. The first and last layers are called input and output layers respectively. In between those layers there can be a number of hidden layers. The type of network shown in Figure 2.3, is called a feed forward network with fully connected nodes [22]. Nodes are never fed backwards in the network and all neurons in one layer are connected to all neurons the following layer. In each neuron a bias is added and an activation function is applied. The output, y, from a neuron can be written as

$$y = f(\boldsymbol{wx} + \theta), \tag{2.12}$$

where $\boldsymbol{x}$ is a vector containing the outputs from the layer before, $\boldsymbol{w}$ is a vector containing the weights for the connections and θ is a bias for each neuron. The function $f(\cdot)$ is called the activation function and maps the weighed inputs to the output of each neuron [22]. A common activation function is the rectified linear unit, *ReLU*, which is a nonlinear activation function [23]. The ReLU function is defined as

$$f(x) = \max(0, x). \tag{2.13}$$

Another common activation function is the hyperbolic tangent function, *tanh*, which maps the output of the node between -1 and 1. An ANN that has many hidden layers is called a deep neural network and the field of deep neural networks is often called deep learning.

2.2.1 Convolutional neural networks

In computers, images are represented as matrices with dimensions $W \times H \times C$, where W is the width, H is the height and C the number of color channels of the image. To use a fully connected ANN with a connection for each input pixel would require an

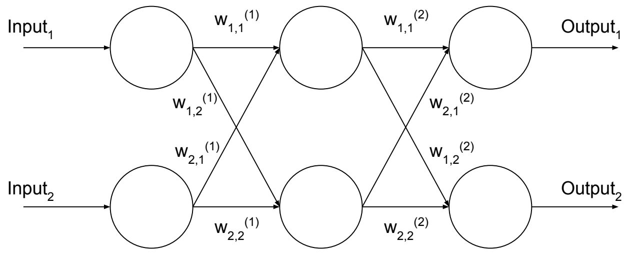


Figure 2.3: An example of a neural network consisting of an input layer, an output layer and one layer in-between called a hidden layer. The connections between the neurons has different weights, denoted w.

abundance of parameters even for a small network. A convolutional neural network (CNN) can be seen as using convolutions between the image and small filters, giving a lot less parameters [5, 24, 25]. Figure 2.4 shows how a CNN with a single filter could look. The image is represented by numbers, and for each time the filter moves one step over the image, it takes multiple values and produces a part of the output. CNNs can be used when the input has a spatial relation, and are commonly used in image analysis applications.

A CNN can also be seen as a number of small networks, each applied to different parts of the image. Because of the spatial relation in the input the networks can share weights. The number of neurons in the networks corresponds to the number of filters used in the convolution, and the networks receptive field to the filter size.

The output from a convolutional layer taking an image, i.e. a matrix, as input will be a new matrix with other dimensions. The number of filters decides the depth of the matrix. The stride, the number of pixels the filter will move in the convolution, and the filter size decides the output height and width. A common way to obtain the wanted height and width after a convolution is to pad the input image with zeros at the borders. The deeper the CNN, the more complex features it can learn [24, 25].

A convolution layer is often followed by an activation function and sometimes a pooling layer. The pooling layer is used as a way of down-sampling. A common pooling function is the max pooling, which gives the maximum output in a rectangular neighborhood. A pooling layer helps to make the output transition invariant, i.e. the output does not change even if the input is slightly shifted [24]. In many applications, a max pooling layer can be replaced by larger stride without loss of accuracy [26].

A variation to a convolutional layer is a deconvolution layer also known as transposed convolutions [27]. These layers are filters that output a larger image than the input shape, and can be used to upsample the input to a larger size.

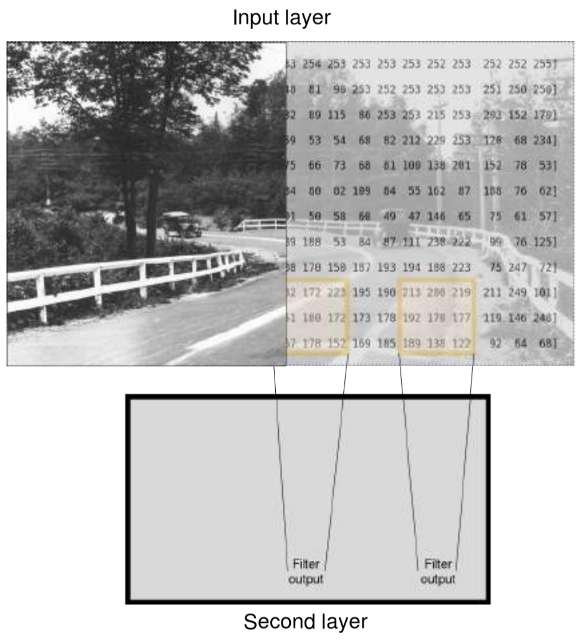


Figure 2.4: A CNN with a single filter applied to an image. The pixels in the image are digitally represented as numbers, which the yellow filter strides over, producing new numbers as output.

2.2.2 Training neural networks

The input and output of a network as well as availability of datasets limits how a network can be trained. There are three different approaches to training a network; supervised, unsupervised and reinforcement learning [28]. These terms have no set definition, and the usage is rather ambiguous in the research community. However, the usage of the terms can still be generalized.

2.2.2.1 Network learning

Supervised learning is used when the dataset contains labels or targets and the network is trained to associate features in the data with the corresponding label or target. An example is to classify a dog or a cat in an image, after training on one dataset of cats and one of dogs.

Unsupervised learning is used when the network is trained to learn properties of the dataset such as distribution or divide it into clusters without any given labels or targets.

Reinforcement learning is when an autonomous agent learns to perform a task without guidance from a human. Instead the agent uses trial and error and learns from the feedback given by the system. This learning method is practical when no correct answer exist at a given time, or when the feedback is delayed.

2.2.2.2 Loss function

The loss, sometimes called error, of what the network predicts is in the supervised case the difference between the ground truth and the prediction. A common way to train a network using supervised or unsupervised learning is by shifting the parameters a small step according to the negative gradient of the produced network loss, which is called gradient decent [28]. The loss function is calculated according to the problem at hand, and could for example be the mean squared error.

2.2.2.3 Optimizers

The objective when training a network is to find the optimal network parameters, i.e weights and biases, of the network to minimize the loss function. Finding the optimal parameters for the network can be a costly process, taking a lot of time and computational power. A number of different algorithms has been developed in recent years that attempts to solve these problems. In deep learning these algorithms are called *optimizers*, and common algorithms are stochastic gradient decent, AdaGrad, RMSProp and Adam [29].

2.2.2.4 Other training methods

Self-supervised learning is a subset of unsupervised learning. A self-supervised network uses the inputs and the predicted outputs to calculate a loss [30]. A set of different loss functions can then be combined to train the network.

A network can be transfer trained or retrained, to fit a new dataset. This is called transfer learning [31]. Retraining a network can done if the new task is similar to how it already is trained.

2.2.3 Further improving a model

A model can still be improved, even after it is trained and has converged for a dataset. There are different techniques to lower the memory size and runtime of a model, and one of these techniques is pruning.

A trained model usually has some weights very close to zero, which do not effect the model output. These weights can be removed, and the neurons without output weights can also be removed, without losing accuracy [32]. The weights and neurons with the smallest effect are pruned and the network is retrained, and the process is repeated until the model loses accuracy. Experiments with well known networks for image classification have shown reduction to 10 % of the original weights [32], reducing the number of weights lowers the runtime. An example of the pruning of a fully connected network can be seen in Figure 2.5.

2.3 Stereo vision using neural networks

A way to calculate high accuracy disparity maps from stereo image pairs is to use CNNs. In this section some of the building blocks commonly used for network design and ways of training a network to produce a disparity map are presented.

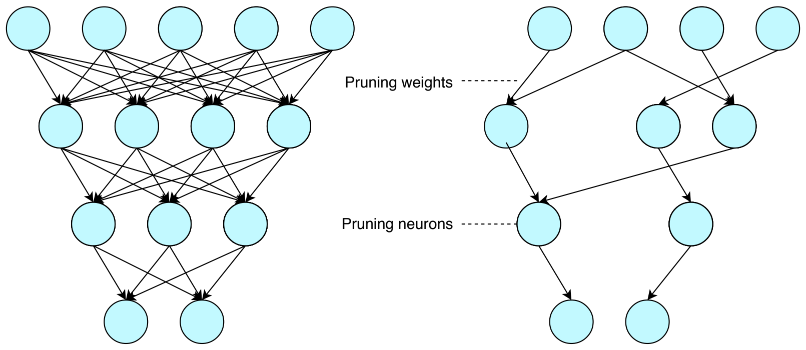


Figure 2.5: The image shows an example of a fully connected network (left) and a resulting network after pruning (right), where both weights and neurons has been removed.

2.3.1 Siamese network

The conventional algorithms used for feature matching can be replaced by CNNs. A common way to extract features and match them using CNNs is by using the *Siamese architecture*, which consists two networks sharing the same weights. The two networks are commonly fed with one image each and provides the matching features as output [33].

2.3.2 Residual connections

When an input to a layer is fed forward and added to the output of a layer further down in the network structure, it is called a residual connection [34]. The output from a residual block can be written as $F(x) + x$, where x is the input to the block and $F(\cdot)$ is the function representing a layer. Residual connections can be used to feed forward information in an encoder-decoder network, as this could make the training easier. It can therefore be used in deeper networks which otherwise would be hard to train.

2.3.3 Reconstruction error

The most common way to train networks to estimate disparity from stereo image pairs is to use supervised learning with ground truth disparity maps. This requires large datasets containing images and corresponding ground truth data. However, only a few large datasets with stereo image pairs and ground truth disparity are publicly available. A way to tackle this problem is to construct the loss function to use just the input and output of the network, no ground truth. In recent publications a so called *reconstruction error* has been used [30, 8]. The disparity between a two images in stereo can be used to warp the left image to the right. The error between the right image and the warped left image can be used as the error on which to train the network, as a perfect disparity map would recreate the other image. The disparity from the left image to the right is written as d^l, and the formula for the reconstructed right image can be denoted as $I_r' = d^l(I_l)$. By using both d^l and the disparity from the right to the left image d^r, the image can be warped twice, with

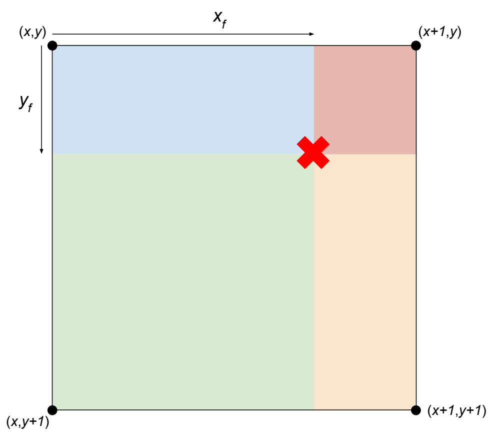


Figure 2.6: A figure to explain how a bilinear sampler works. The four neighbours of the the red cross are weighted together to form a value. In the warped image that value is then given to the pixel the cross has transformed to. The x_f and y_f represent the decimal pixel coordinates of the cross in the original image.

both output disparity maps, producing itself as $I_l'' = d^r(d^l(I_l))$. This kinds of errors are both used in [8] and [30]. A visual representation of how I' is created can be seen in Section 3.3.3.

2.3.3.1 Bilinear sampling

Image warping is a transformation that maps all the pixel in the input image to positions in a new image [35]. A common linear transformation is the affine transformation, given as:

$$\begin{aligned} u &= a_0x + a_1y + a_2, \\ v &= b_0x + b_1y + b_2, \end{aligned} \tag{2.14}$$

where x and y are the old pixel positions, u and v are the pixel positions in the new image and a and b is the transformation in question [35]. Often, the resulting values for u and v are not integers. This can be solved by rounding the values to integers but result in loss of information, or by using *bilinear sampling*. The bilinear sampling method is done by calculating from which location in the old image the value should be taken for a specific pixel location in the new, warped image. The value is then calculated by the weighted sum of the four neighbouring pixels to the location [36]. A warping $I_{\text{warp}}[u,v] = I[x_f, y_f]$ where x_f and y_f is the decimal values of the location where to obtain the pixel value is illustrated in Figure 2.6. The sampled value for the warped image is then:

$$\begin{aligned} I'[u,v] = (x+1-x_f)(y+1-y_f)I[x,y] + (x_f-x)(y+1-y_f)I[x+1,y] + \\ (x+1-x_f)(y_f-y)I[x,y+1] + (x_f-x)(y_f-y)I[x+1,y+1]. \end{aligned} \tag{2.15}$$

In [37] a network using a fully differentiable implementation of bilinear sampling is presented. This implementation is later used for implementations of reconstruction error for image warping using the disparity [30, 8].

2.3.3.2 Structural similarity between images

A problem often occurring when working with images is how to decide how similar two images are. A simple solution is to calculate the element-wise norm between the images, however that is not always a suitable way to calculate similarity. A structural similarity measure, SSIM, that calculates the similarity of the luminance, contrast and structure for images was presented in [38]. The function, SSIM, for structural similarity between two images, I and I' is defined as:

$$\mathrm{SSIM}(I, I') = [l(I, I')]^{\alpha} \cdot [c(I, I')]^{\beta} \cdot [s(I, I')]^{\gamma}, \tag{2.16}$$

where $l(\cdot)$, $c(\cdot)$ and $s(\cdot)$ are functions for comparing luminance, contrast and structure respectively, and the α, β and γ are weights of these functions. In [38] the SSIM-function is simplified to:

$$\mathrm{SSIM}(I, I') = \frac{(2\mu_I \mu_{I'} + C_1)(2\sigma_{II'} + C_2)}{(\mu_I^2 + \mu_{I'}^2 + C_1)(\sigma_I^2 + \sigma_{I'}^2 + C_2)}, \tag{2.17}$$

where μ is the mean of the image, σ the standard deviation and C_1 and C_2 are constants defined as:

$$C_1 = (K_1 L)^2, \tag{2.18}$$
$$C_2 = (K_2 L)^2, \tag{2.19}$$

and $K_1, K_2 \ll 1$ and L is the dynamic range of pixel values [38]. Using this measure of similarity can be helpful when comparing images, as seen in both [30] and [8].

2.3.4 Recent research

A common indicator of the accuracy and performance of different ANNs, is the Kitti benchmark suite [39]. The benchmarks on Kitti are limited to a few categories which are evaluated on publicly available datasets.

One network using the reconstruction error for unsupervised, or rather self-supervised training is the SsSMnet [30]. It ranked as the fifth best network when it was published to the Kitti stereo 2015 benchmark, and was more accurate than many nets using supervised learning.

A network that has interesting properties is a CNN for LiDAR-Camera Fusion [40]. The paper explores the different approaches to the fusion between two input sources. The paper introduces the novel fusion strategy called cross-fusion, which connects the two branches of data and enables the network to integrate multimodal information at any abstract level.

2.3.4.1 The geometry and context network

A network scoring high on the Kitti benchmark scoreboard is the Geometry and context network, called the GC-net [6]. The GC-net consists of two Siamese networks of 2D convolutions used for feature extraction, combined with a cost volume calculation. The cost volume calculation is a function that calculates the cost of matching each pixel in one of the input images with all other pixels in the other image horizontally. This cost is combined with the output matrices from the Siamese networks, giving a 4D volume. The cost computation is then refined using multiple 3D convolutions. Following this is a soft argmin activation function which is differentiantiable, and defined as:

$$\text{argmin} \sum_{d=0}^{d_{\text{max}}} d\sigma(-c_d), \tag{2.20}$$

where d is the disparity, d_{max} the maximum allowed disparity, $\sigma(\cdot)$ the argmax function and c the predicted cost [6, 30]. The GC-net has an outlier percentage of 2.87 % on the Kitti scoreboard and is top ranking.

2.4 Applying the theory

The theory which has been presented in this chapter is the basis for Chapter 3, and is used to complete the goal of creating a full HD disparity map at high frequency.

The stereo vision theory, Section 2.1, was used in the selection and creation of datasets. The theory which combines neural networks and stereo vision, Section 2.3, was used in order to design the networks. The different building blocks of neural networks was combined to create new network structures while the reconstruction error in Section 2.3.3 allowed for self-supervision in the networks. Section 2.3.4, Recent research, was used as an initial inspiration for network structures.

3 Method

In this chapter the method and work flow used in the master's thesis are described. Since the hardware was provided by CPAC, the main focus for this thesis was software and more specifically to design a robust network that can be used in a real world application. The work was carried out iteratively, where network was improved according to the evaluation of the previous iteration. This chapter describes the hardware used in the project, the datasets used for network training and the process of creating a network which fulfills the goals. This chapter also include immediate results from the networks used to further iterate the design.

3.1 Correlation between disparity, depth and hardware

The correlation between disparity and depth depends on the specific cameras and how they are positioned. More specifically the baseline, the focal length and the pixel size is needed to calculate a correlation. The disparity estimations can be performed on-board the camera rig using the NVIDIA Jetson TX2 embedded computing device [41]. The TX2 is built around a GPU; a NVIDIA Pascal™ with 256 NVIDIA CUDA® cores and a total RAM of 8 GB. The camera rig consists of four cameras mounted on a distance 14 cm apart. The cameras are from Leopard Imaging Inc [42]. They are able to capture images in full HD at 60 fps, have a focal length of 5 mm and a pixel size of 3.75 µm. A photograph of the camera rig can be seen in Figure 3.1.

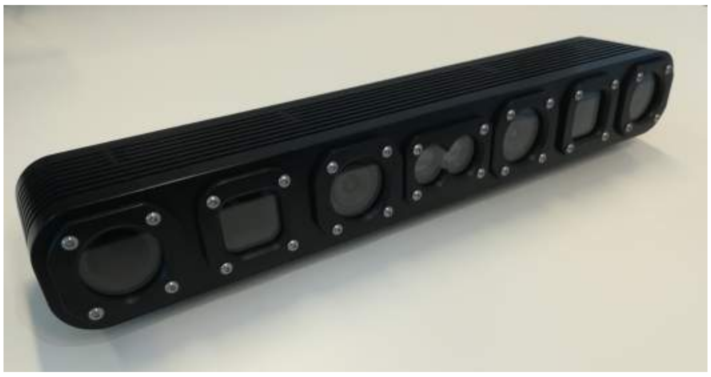


Figure 3.1: The camera rig consisting of four cameras and a laser rangefinder.

Using Equation 2.11 and the camera specifications for the rig, the correlation between disparity and depth for the different camera pairs can be visualized. The resulting graph can be seen in Figure 3.2. Objects closer than 0.5 meters to the camera will have a disparity of more than 300 pixels for the camera pair with the shortest baseline. For the cameras further apart the disparity will be even higher. Some algorithms and networks require a maximum allowed disparity, called $d_{\max}$, for example the GC-net described in Section 3.3.1. For the GC-net, the $d_{\max}$ decides the interval of which to calculate the matching pixels. With a maximum allowed disparity of 200 pixels (px), and a baseline of 14 cm, the depth of an object slightly closer than one meter from the camera pair can be estimated. Apart from the cameras, the rig is also equipped with a one point laser rangefinder, LiDAR, that measure distances between 0 and 100 meters [43]. The LiDAR is not used in this project.

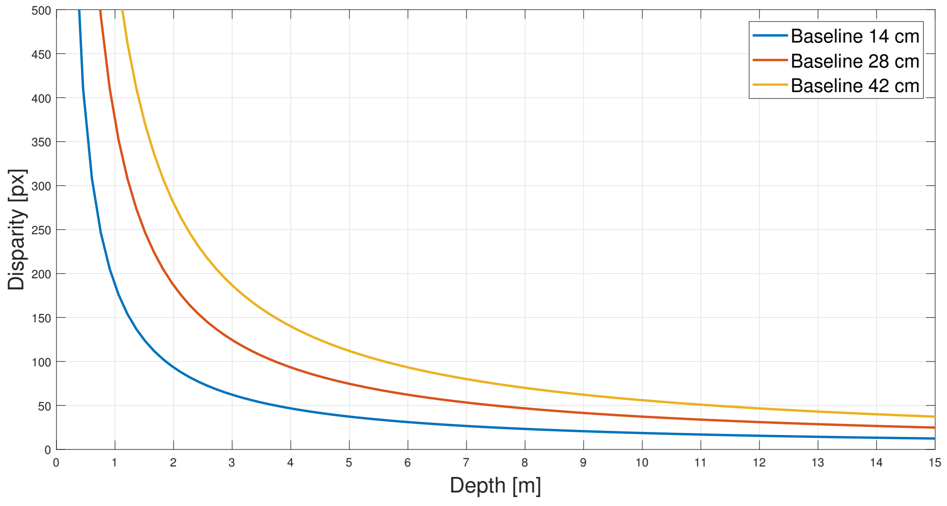


Figure 3.2: A graph of the correlation between disparity and depth for the different camera baselines. As seen a disparity of 200 pixels for a baseline of 14 cm corresponds to a distance of a little less than one meter.

3.2 Dataset selection

In this section the different datasets used for network training and evaluation are presented. Most of the data used is synthetic, meaning that it is generated in a virtual world and has a dense ground truth. Real-life ground truth disparity data is often sparse, as it is commonly generated from LiDAR data. One can make a difference between publicly available datasets, generated by others, and datasets generated as a complement for the purpose of this project alone. The datasets generated for this project are created to match the hardware specifications of the rig, e.g. the camera positioning and the camera parameters.

3.2.1 Available datasets

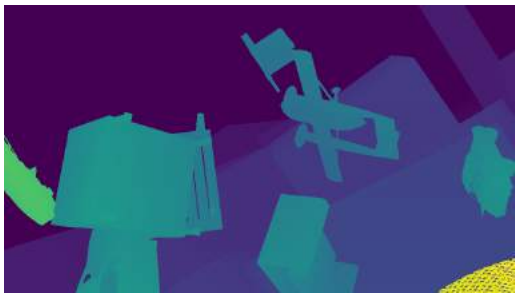
There are many publicly available datasets for different purposes, however many of those are not applicable for disparity estimations. By getting an overview of relevant datasets, it is possible to decide upon what needs to be created.

3.2.1.1 FlyingThings3D

The FlyingThings3D (FT3D) dataset [44] is a simulated dataset consisting of a stereo image pair with ground truth left and right disparity. It is a total of 21 818 frames training data and 4 248 frames test data in the dataset. The resolution of the RGB images are 960×540 pixels. As the name suggests, the dataset consists of images of different objects flying around in a randomly generated environment. Some of the objects are so close that they occlude the view from one of the cameras. To tackle this problem, data containing a disparity higher than 200 px was sorted out of the dataset. Since the FT3D data is randomly generated, the data has low bias. An example of the FT3D data can be found in Figure 3.3.

(a) Left image

(b) Right image


(c) Ground truth left disparity. Brighter color means higher disparity.

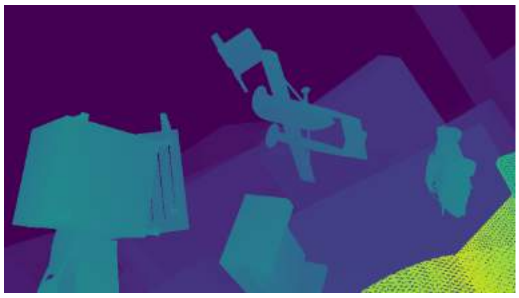
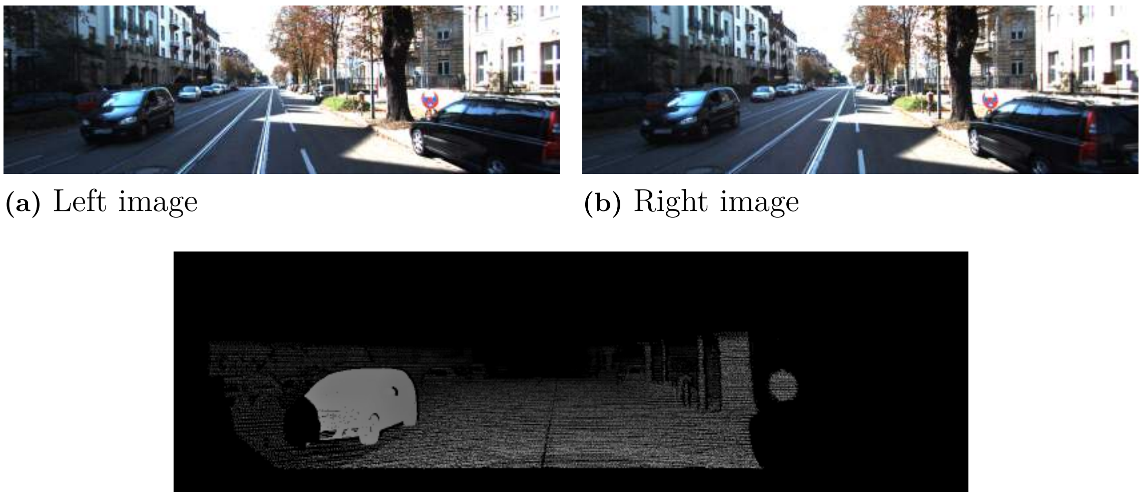

(d) Ground truth right disparity. Brighter color means higher disparity.

Figure 3.3: An example of how an image pair and the resulting disparities look in the FT3D dataset.

3.2.1.2 Kitti Stereo 2015

Kitti is a project from Karlsruhe Institute of Technology and Toyota Technological Institute in Chicago with the goal of producing real world benchmarks for computer vision [39]. The Kitti Stereo 2015 dataset contains two images captured in stereo and the ground truth disparity map for the left images, which is based on LiDAR data points. The ground truth data is sparse and contains only data points close to the cameras. The dataset has a resolution of approximately 1242×375 pixels, though the size vary throughout the dataset. There are only 400 image pairs with ground truth disparity, however 8400 image pairs without ground truth exist. An example of the Kitti data can be found in Figure 3.4.


(a) Left image

(b) Right image

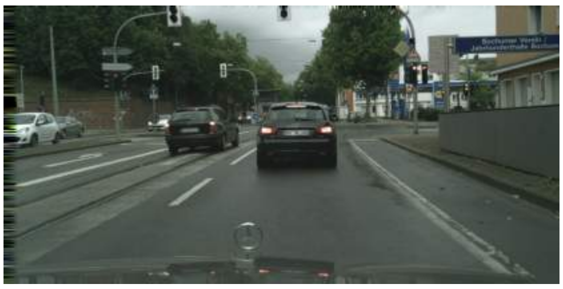
(c) Ground truth left disparity. A brighter point means a higher disparity. Occluded or uncertain points has been removed.

Figure 3.4: An example of how an image pair and the resulting left disparity look in the Kitti dataset.

3.2.1.3 Cityscapes

Cityscapes is large scale real life dataset with two images captured in stereo taken from 50 different cities in Germany [45]. The Cityscapes dataset consists of approximately 25,000 image pairs with sparse ground truth disparity data. The images have a resolution of 2048×1024, however the right image contains some errors along the left and top side of the image. These errors are visible in Figure 3.5b, where the left edge of the image looks like it has been padded. Also, all the images contain the hood of the car with a Mercedes-Benz star. The images were cropped to get rid of the errors along the edges as well as the hood of the car which would otherwise generate a constant bias. The Mercedes-Benz star was left in the images, since cropping it away would lead to loosing too much data. Worth to note about the Cityscapes dataset is that the ground truth disparity sometimes contains errors and the results obtained on this dataset might be slightly misinforming. An example of the Cityscapes data can be found in Figure 3.5.

(a) Left image


(b) Right image

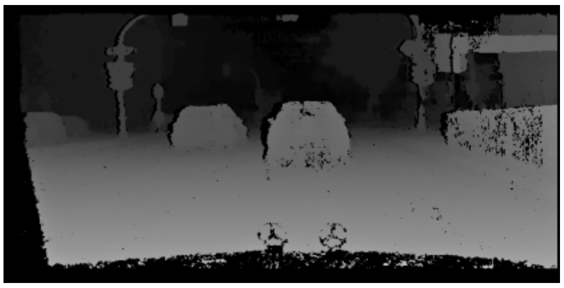

(c) Ground truth left disparity. A brighter point means a higher disparity.

Figure 3.5: An example of how an image pair and the resulting left disparity look in the Cityscapes dataset. The left and the top side of the images might contain errors, in these image pair seen most in the right image.

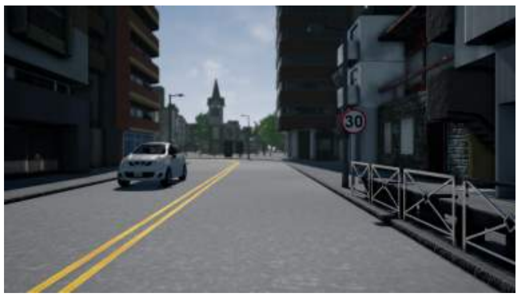
3.2.2 Generated data

To have a dataset with correct hardware parameters to train a network, a simulated dataset was generated, using the tool Carla Driving Simulator [46]. Carla is an open source driving simulator developed for autonomous drive research on the Unreal Engine platform which allows for custom camera parameters like resolution, FOV, position and rotation. To make the simulated data similar to the data that can be obtained from the camera rig, the simulation was done with four cameras aligned 14 cm apart and a depth camera. With the depth camera, a ground truth disparity map could be calculated for each camera pair using Equation 2.11 and the camera specifications in Section 3.1. The result was three left disparity maps, one for each camera pair with different baselines. The horizontal FOV was set in Carla, so the focal length of the cameras on the rig and in the generated dataset are the same. The focal lengths of the cameras on the rig are 5 mm and the pixel size 3.75 µm. Equation 2.1 could then be used to calculate the horizontal FOV to 71.508° as

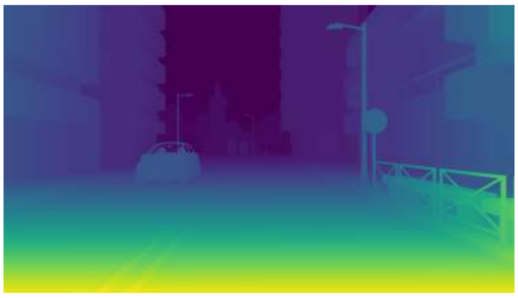
$$\text{FOV}_{\text{horizontal}} = 2 \arctan\left(\frac{1920 \cdot 3.75 \cdot 10^{-3}}{2 \cdot 5}\right) = 71.508. \tag{3.1}$$

An example of how the Carla data looks can be found in Figure 3.6. Another dataset was captured in Gothenburg city as a proof of concept. This is presented in Section 3.4.

(a) Left image



(b) Right image



(c) Ground truth left disparity. Lighter color means higher disparity.

Figure 3.6: An example of how a left and right image pair and the resulting left disparity look like in the simulated Carla data.

3.3 Neural network

The goal is to design a network that takes high resolution stereo image pairs as input and outputs high accuracy disparity maps, at multiple samples per second. Different network designs were created, trained and evaluated. New networks were designed based on the conclusions drawn in the evaluation. This process was carried out iteratively until a final net was acquired. The software used for the network design and evaluation was Keras [47] with a TensorFlow [48] backend.

The initial designs were inspired from high performing networks on the Kitti benchmark. The later network designs combined building blocks from recently published networks to achieve new designs. All the networks were designed to use a single stereo camera pair. How to take advantage of the full potential of the camera rig is discussed in Section 4.3.

To train a network, either images with a corresponding ground truth data or a self-supervised learning approach is required. The initial goal was to design a network that could be used to predict satisfying results using supervised learning, since it is easier to implement and test a network using supervised learning. When a well working network was obtained, the focus could shift to how self-supervised training could be used in this context, to enable the network to learn new data and environments without corresponding ground truth data. With a self-supervised approach, the network could be used in a real-life application where ground truth disparity does not exist. All training was done with a GeForce GTX 1080 Ti which has 3584 CUDA cores, 14 times as many as the TX2.

For evaluating the network and comparing the results, evaluation methods and metrics had to be decided upon and implemented. The networks were evaluated with regards to accuracy, runtime and also computational power and memory usage, to decide whether it will work on board the camera rig.

When evaluating the different networks between iterations the Kitti benchmark error was used. The Kitti error is calculated by how many pixels that can be labeled as outliers:

$$\text{Outliers} = (|d_{\text{GT}} - d_{\text{est}}| > \tau_0) \cap \left(\frac{|d_{\text{GT}} - d_{\text{est}}|}{|d_{\text{GT}}|} > \tau_1 \right), \tag{3.2}$$

where τ_0 and τ_1 are thresholds, for Kitti $\tau_0 = 3$ and $\tau_1 = 0.05$. The d_{GT} is the ground truth disparity and d_{est} the disparity estimated by the network. If the pixel fulfills both parts of the intersection, the pixel is labeled as an outlier. The number of outliers is only calculated for pixels where a ground truth depth exists. Changing τ_0 to higher and lower values gives good indication on how the errors are distributed. Another way the prediction error can be calculated is by using the absolute relative error,

$$E_{\text{abs\_rel}} = \frac{1}{N} \sum_{N} \frac{|d_{\text{GT}} - d_{\text{est}}|}{|d_{\text{GT}}|}, \tag{3.3}$$

and the squared absolute relative error,

$$E_{\text{sqr\_rel}} = \frac{1}{N} \sum_{N} \frac{|d_{\text{GT}} - d_{\text{est}}|^2}{|d_{\text{GT}}|}. \tag{3.4}$$

The common root mean squared error, defined as

$$E_{\text{RMS}} = \sqrt{\frac{1}{N} \sum_{N} (d_{\text{GT}} - d_{\text{est}})^2}, \tag{3.5}$$

can also be used as well as the mean absolute error, also called endpoint error [49], EPE,

$$E_{\text{EP}} = \frac{1}{N} \sum_{N} |d_{\text{GT}} - d_{\text{est}}|. \tag{3.6}$$

3.3.1 GC-network

For the first network design, we used the structure of one of the top performing models on the Kitti benchmark. The network chosen initially was the Geometry and context network, called the GC-net [6], described in Section 2.3.4.1. In this section the design, training configuration and some results are presented together with an evaluation of the network. The evaluation part contains the conclusions drawn on why this network does not fulfill the goals, and how to address that problem.

3.3.1.1 Network design

The network was designed using the main ideas from the GC-net, but reducing the network size to allow for quicker testing and evaluation. This lighter version designed is called GC-lite. The GC-lite network consists of two Siamese nets using shared weights to extract a total of 32 filters in each image. The outputs from the Siamese networks are combined using the cost volume calculation. The output from the cost volume is passed through five 3D-convolutional and two 3D-deconvolutional layers. Before the deconvolutional layers a total of 64 filters are extracted. The last layer contains the soft argmin activation function. The output from the network is the left disparity map.

3.3.1.2 Training

To train the network the input images were cropped at random to the size of 256 px height and 512 px width and normalized between -1 and 1 for all three color channels respectively. The loss function used was the mean absolute error and the network was trained using RMSprop. When using the FT3D dataset the d_{max} was set to 200 px, since all images in the dataset containing higher disparities have been removed as outliers. Since the network uses a cost volume, the ground truth disparity used for training was not normalized. After the model was trained for more than 90 hours, the loss had not converged, but was declining very slowly.

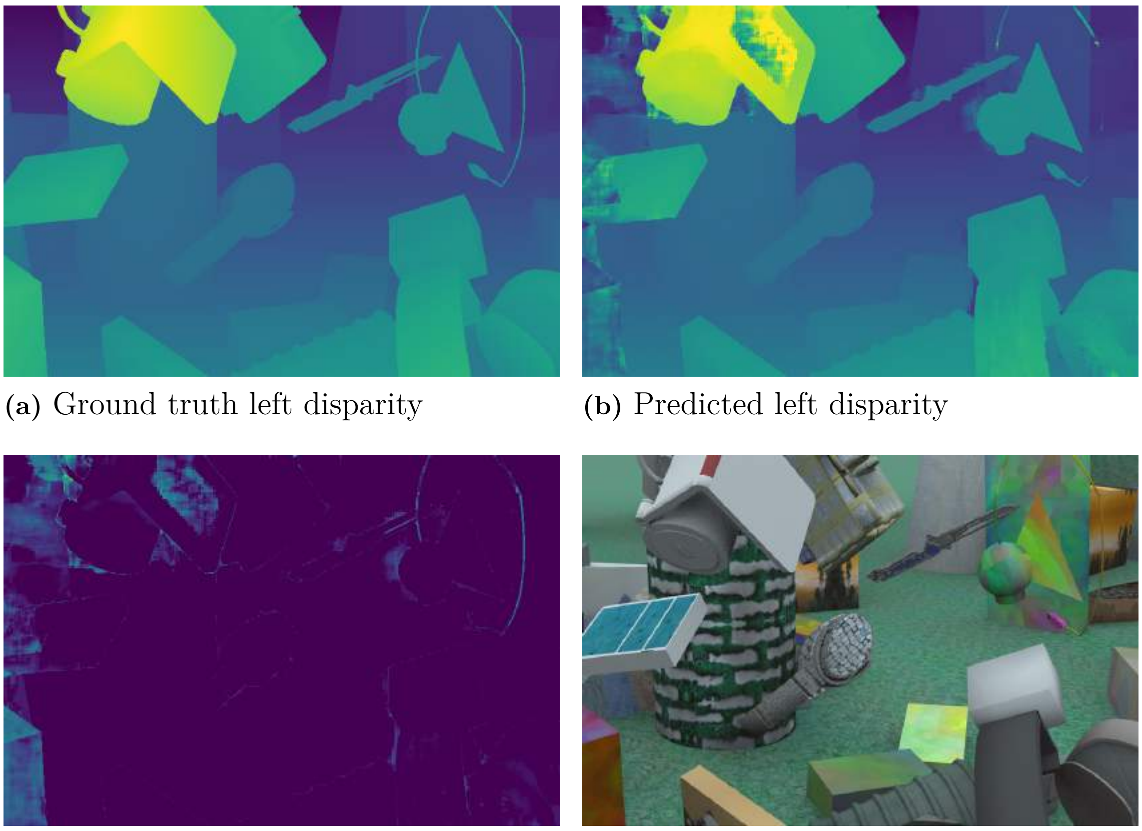
3.3.1.3 Results

When trying to predict a high resolution disparity map using the GC-lite network a GeForce GTX 1080 Ti with 11 GB of memory runs out of memory. One of the largest resolutions this GPU can handle is 768×512 px. Using this image size, the average predicting time on the GPU is 0.96 seconds. An example of how the left disparity map looks when predicting on the FT3D dataset is found in Figure 3.7 together with the left input image and ground truth disparity map. The network was not optimized with regard to accuracy as it could be concluded that the network would not be able to fulfill the goal presented in Section 1.1.1; it is a too computationally and memory heavy design. The numerical results can be found together with a comparison of different designs in Section 4.1.

3.3.1.4 Evaluation

A conclusion that can be drawn from the result is that the network design is too memory heavy for the TX2, since the target resolution can not even be run on the GeForce GTX 1080 Ti. The way the GC-lite is built, using the cost volume followed by the 3D convolutions and deconvolutions is even too computationally heavy for the TX2.

Due to the computation and memory issues, higher dimensional volumes can not be used. The designs on the Kitti benchmark are graded on accuracy, and they might not be optimized for computation speeds. New approaches should therefore focus on lightweight building blocks and efficient network structures in order to fulfill the goals, and not only accuracy.


(a) Ground truth left disparity (b) Predicted left disparity

(c) Outlier map (d) Left input image

Figure 3.7: Results from the GC-lite network. It can be seen that the network has problems with occluded and textureless areas, as well as the left hand side of the image. The outlier map shows the errors as an intensity map where the yellow part, highest intensity, corresponds to an absolute error 91 pixels. The number of outliers in this prediction is 11.44 %.

3.3.2 Cross-CNN

A lightweight network was designed with inspiration from the encoder-decoder structure commonly used in CNNs for semantic segmentation, and the cross-fusion network presented in [40]. This kind of network structure, containing only 2D convolutions and early compression of the image, creates a network which is more lightweight than the GC-lite network. A concept image of the network design can be found in Figure 3.8. The first design of the network was tested with a single output disparity map, and the immediate results proved the design had a fast runtime with potentially high quality results. Using the initial network and tuning design parameters, different approaches to the network was compared, and a final design was chosen.

3.3.2.1 Network design

The network uses the common encoder-decoder structure which quickly reduces the size of the two input images. In order to keep important information given in the input images, the feature size is increased when the height and width are reduced.

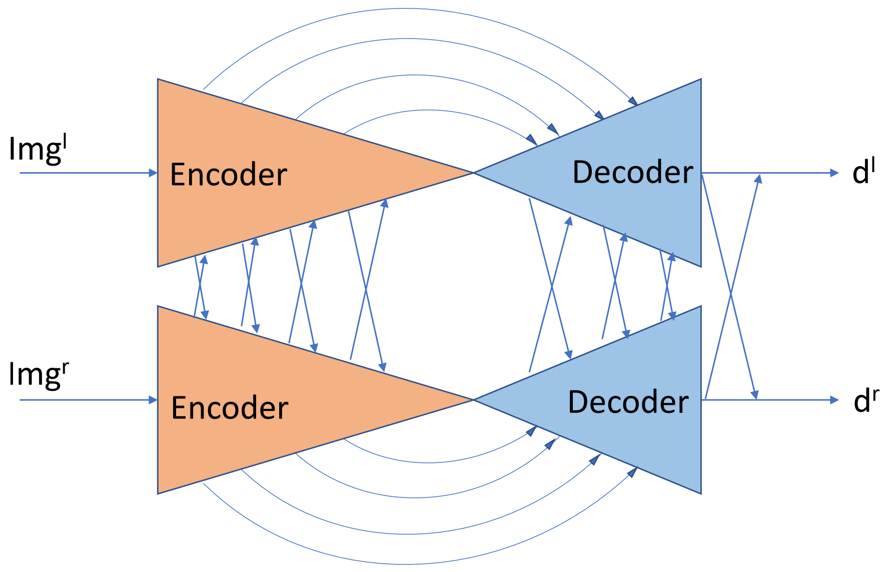


Figure 3.8: A concept image of the design of the cross connected-CNN. The two networks share the same weights and are connected with cross connections that has trainable scalar weights. Each arrow in the image represent an output from a layer added to the input of another layer in the arrows direction. The arrows between the networks represents cross connections and the arrows from the encoder to the decoder residual connections. The left image is fed to the part of the network producing the left disparity map, and vice versa.

A detailed figure of the cross connected-CNN (X-CNN) can be seen in Figure 3.9. The network consists of residual connections between the encoder and decoder which feed important information forward in the network and ease the training. In order to combine the two Siamese networks continuously, connections between the networks with trainable scalar values are used on each layer, called cross connections. The network was initially designed to give the left disparity map as output for quick testing of the network. The network was later redesigned to output both disparity maps.

3.3.2.1.1 Building blocks of the X-CNN The initial X-CNN was built up with convolutions to lower the input size quickly, followed by the same amount of deconvolutions. It had multiple residual connections, and every layer also had a cross connection with trainable weights. The initial network scaled down the input image to $\frac{1}{64}$ of the input size. It contained 20 layers evenly distributed in the encoder and decoder with residual connections, and a final layer to calculate a single disparity map.

To design a final version of the X-CNN, the effect of the different design parameters were investigated by designing new networks and changing only a few parameters at the time. The different designs were trained for 4 epochs each on the Carla data and their losses were compared to the initial X-CNN.

The depth of the network and number of layers was tuned to find a quick, but

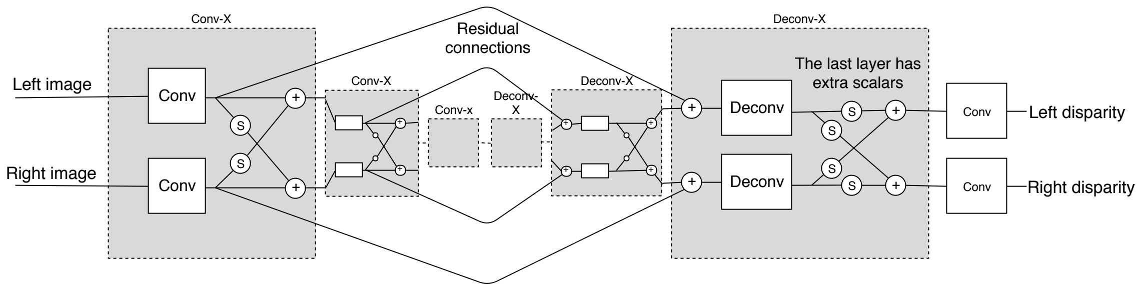


Figure 3.9: An overview of the building blocks in the cross connected-CNN network. Like an encoder-decoder structured CNN, the input is sampled down and then up to the size of the input image again. Between some of the encoder and decoder layers are residual connections. After each convolution, the outputs from the same layers are combined with a cross connection with trainable scalar weights.

accurate network. To compare depth and runtime a three layer network using 2D convolutions and deconvolutions on full HD images was compared to a more complex network. As seen in Table 3.1, the number of layers have an effect on the runtime but not to such extent that it has to be minimized. The prediction time and the input size is linearly correlated and any network predicting on full HD images will not be as fast as needed. The focus was therefore to obtain good accuracy with the 2D convolution and deconvolution structure.

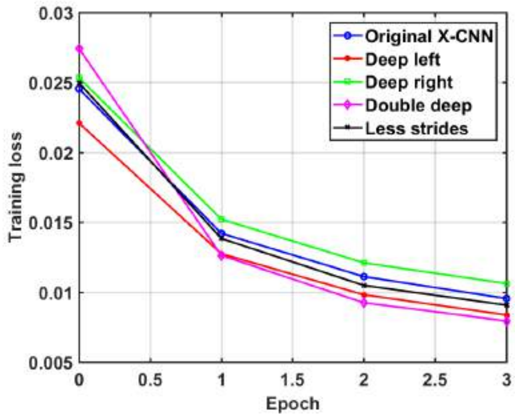
Table 3.1: Runtimes for predicting a disparity map, with different image and network sizes, on a GeForce GTX 1080 Ti. The networks consists of mirrored convolutions and deconvolutions.

Image resolution [px]	21 layers	3 layers
1920×1024	0.201s	0.136 s
960×512	0.043 s	0.030 s

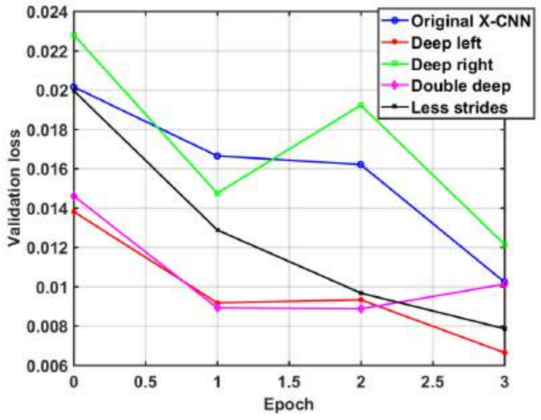
Networks with deeper encoder (deep left), deeper decoder (deep right) and a combination of both (double deep) was tested. Other networks was designed to test the effect of the residual connections (in figure called skips) on every other layera and on each layer, cross connections, number and size of filters. The training results can be found in Figure 3.10. From the different training and validation losses, new networks with a combination of these approaches could be designed.

The second round of designs used was a combination of less sub-sampling than the original, fewer filters in each layer, certain residual connections and different combinations of the number of layers. A few designs of different depths, number of layers, were trained again and the training and validation losses can be seen in Figure 3.11. From these the final network was chosen.

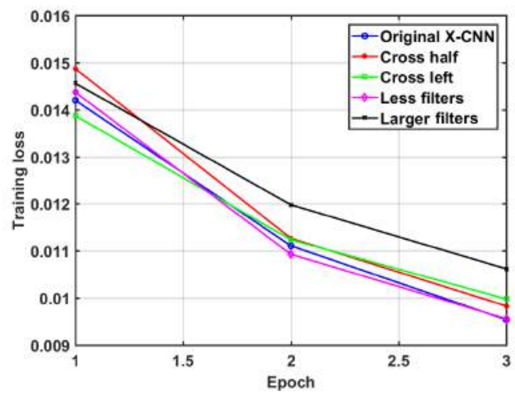
3.3.2.1.2 Final X-CNN design The final network was designed after the previous designs were trained and evaluated, and it is very similar to the initial design. It has 15 layers in the encoder, 10 in the decoder and a final convolutional layer with only one filter. The input images are scaled down to a size a $\frac{1}{32}$ of the original



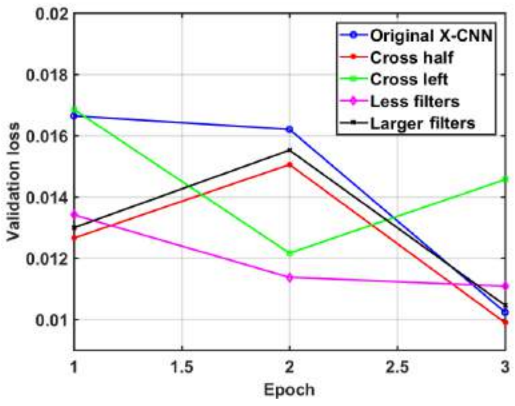
(a) Training loss for networks of different depths



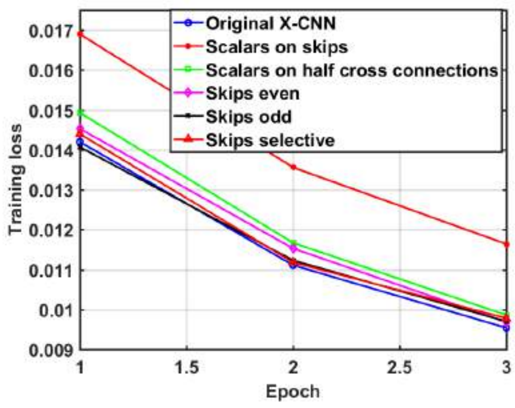
(b) Validation loss for networks of different depths



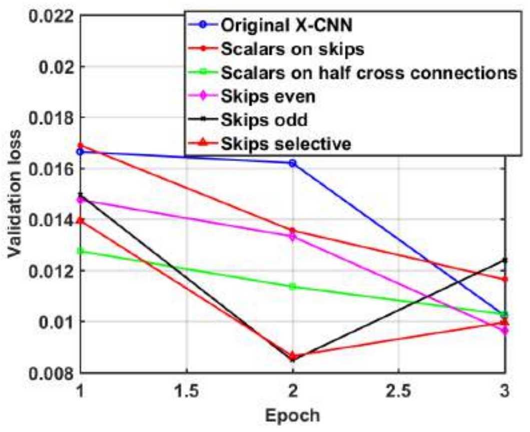
(c) Training loss for networks containing different cross connections and filters



(d) Validation loss for networks containing different cross connections and filters

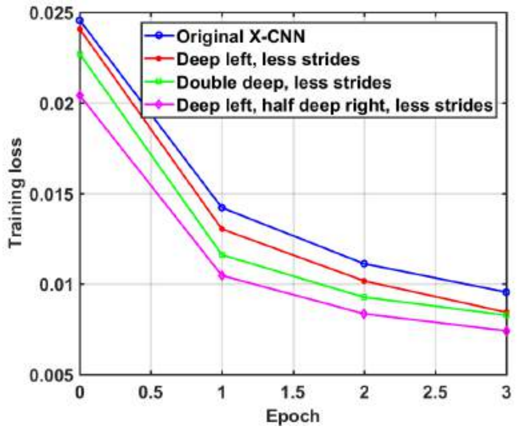


(e) Training loss for different residual, or skip connections, and scalars

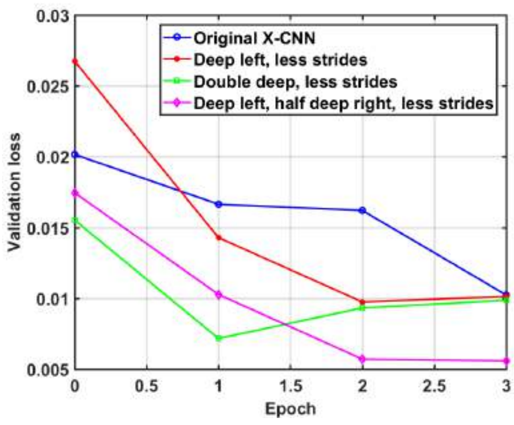


(f) Validation loss for different residual, or skip connections, and scalars

Figure 3.10: Results from different design approaches of X-CNN trained on Carla data. As seen the epoch axis stars at 0, meaning it has already been trained for one epoch. In some of the plots the first epoch is cropped away, for easier viewing.



(a) Residual connections, and scalars training



(b) Residual connections, and scalars validation

Figure 3.11: Results from different designs of X-CNN trained on Carla data, the second iteration. The epoch number 0 corresponds to one epoch of training. These results were used to design a final X-CNN.

size through convolutions. The number of filters for each layer range between 20 and 80, with the most filters in the middle layers. The network has some residual connections and cross connections between each layer of the Siamese networks. The output is a left and a right disparity map with the same size as the input image. A table with a description of every layer in the final X-CNN is found in Appendix A.

3.3.2.2 Training

The final X-CNN was trained on different datasets to compare the accuracy and how it handles different challenges. The optimizer used was Adam, as it showed better results than RMSprop. The network was trained on the Carla data to see how it handles large images, and on FT3D data to see that the network can perform well on data with large disparities that is not as biased as the Carla data. Both datasets have a ground truth disparity, and the loss is calculated as the absolute difference between the estimate and the ground truth.

As the network scale down the input image to $\frac{1}{32}$ of the input size, the images needs to be cropped or padded to be evenly dividable by 32. When training the network on the Carla data, the data was randomly cropped to different sizes for each batch. Randomly cropping the image should also make the network more robust and not be affected as much by the bias in the dataset. The smallest crop size used was 512×320 and the largest 1920×1024 px.

3.3.2.3 Results

A network design which only use 2D convolutions and deconvolutions can be fed with Full HD images without requiring too large memory resources. The average prediction time on full HD images, calculated as a mean value over 100 samples of the Carla test data in full HD, using a GeForce GTX 1080 Ti was 0.2251 seconds.

A network for predicting on Carla was trained for 50 hours in total and the average number of outliers, calculated with Equation 3.2, was 2.16 %. An example of how the network predicts on the Carla data, together with an intensity map of the outliers can be found in Figure 3.12.

A network for predicting on the FT3D dataset was trained for 50 hours on the whole training dataset. The average prediction time, over 100 samples of the test data, was 0.0524 seconds and the average number of outliers, calculated using Equation 3.2 is 9.65 %. An example of how the network predicts on the test data can also be found in Figure 3.12.

3.3.2.4 Evaluation

From the early results of predicting using the X-CNN, it can be stated that the network design will not fulfill the goal of a prediction frequency of 10 Hz on the TX2. As seen in Table 3.1, when using a three layer to predict on full HD images, the prediction frequency on the GeForce GTX 1080 Ti is approximately 7 Hz, and that is a much more powerful GPU than the TX2. The prediction time is linearly correlated with the input size, seen in Table 3.1, and no CNN fed with two full HD color images can predict at 10 Hz. The goal of 10 Hz prediction on full HD images on the TX2 will therefore not be fulfilled. The X-CNN does not perform exceptionally well on datasets with large disparities and large occluded areas such as the FT3D dataset. For the Carla dataset however, the network obtains very accurate results. A table with more results can be found in Section 4.1.

The X-CNN was chosen as a good base for further designs as it is relatively lightweight with a fast runtime. The network showed good accuracy for the Carla dataset, but could only be trained on annotated datasets.

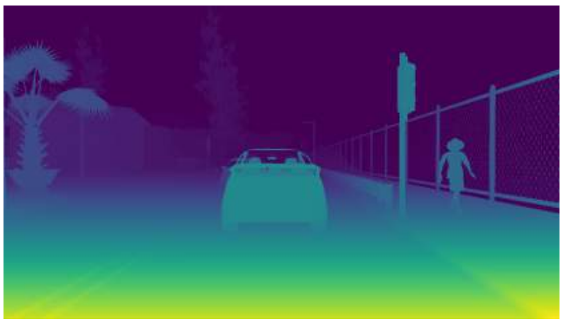
3.3.3 Self-supervised Cross-CNN

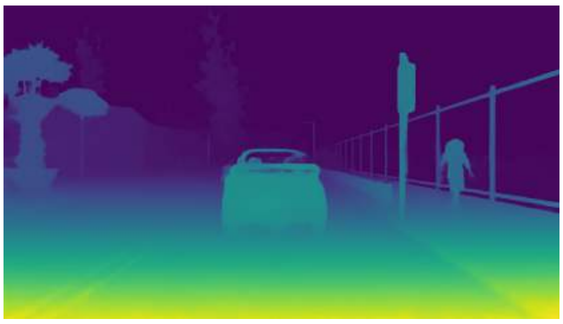
A network which can be trained with a dataset of only stereo image pairs can easily be applied to new environments and can be very useful in a real life product. For the self-supervised approach the loss function of the network had to be rewritten. The new loss function was a compilation of different calculations using only the inputs to the network and the predicted disparities.

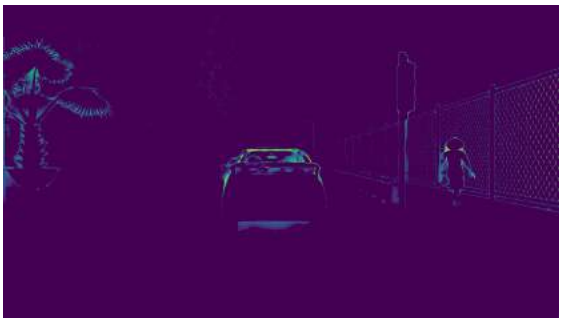
3.3.3.1 Network design

For the self-supervised approach the X-CNN designed to output both the left and right disparity map was used, as both are needed for the self-supervised loss computation, described later in this section. Furthermore, to allow the different kinds of self-supervised loss functions presented in this section a denormalization of the disparity was needed inside the network, as the bilinear sampling in the warping function needs the disparity to not be normalized. The denormalization of the disparity, d_{out} in the network is done by:

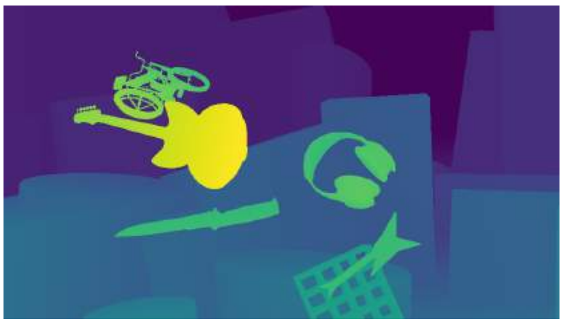
$$d_{notnorm} = 100(d_{out} + 1), \tag{3.7}$$

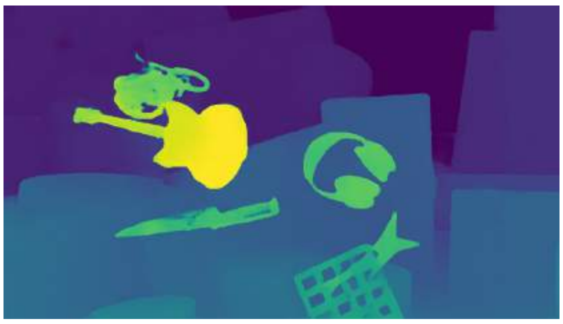

(a) Ground truth left disparity

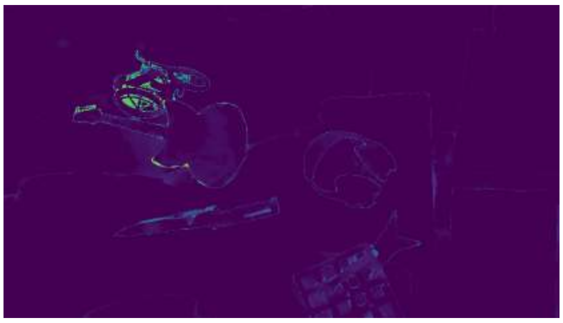

(b) Predicted left disparity


(c) Outlier map, with 3.38 % outliers and max absolute error is 28 px

(d) Left input image


(e) Ground truth left disparity


(f) Predicted left disparity


(g) Outlier map, with 8.99 % outliers and the max absolute error is 117 px

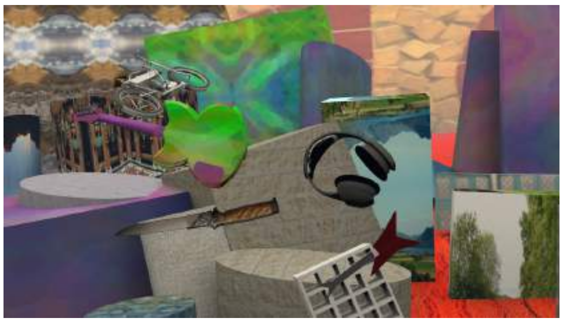

(h) Left input image

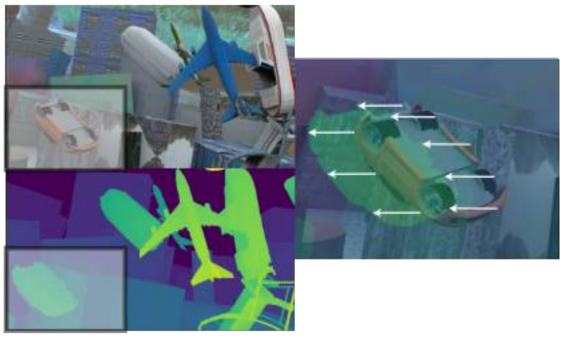
Figure 3.12: Results from the X-CNN network trained on Carla, image (a)-(d), and FT3D, image (e)-(h), respectively. The network does not perform well on edges and occluded areas. The outlier map is a heat map for all the pixels labeled as outliers.

giving a denormalized disparity, $d_{notnorm}$ between 0 and 200 for a d_{out} between -1 and 1. The denormalization wass done to output disparities between 0 and 200, because it complies with the discussion in Section 3.1, that 200 is a sufficient disparity for cameras with the closest baseline.

For the self-supervised network training, a loss function which is not using the ground truth had to be implemented. This loss function can be a combination of different self-supervised loss functions weighted together. The main loss functions used are presented in [8] and [30]. The main idea for the self-supervised loss function is to find the reconstruction error by warping one of the images to the other by using bilinear sampling, see Section 2.3.3.1, and the predicted disparity maps. Since the disparity map is only a measure of horizontal disparity, no vertical transformation is done when warping the images. How similar the warped image and the real image are can be computed in different ways. A simple way is to calculate the norm between the images;

$$L_{\text{norm}} = |I - I'|, \tag{3.8}$$

where I corresponds to one of the images the network takes as input and I' the other input image warped, which was further explained in Section 2.3.3. A visual representation of how I' is generated can be seen in Figure 3.13. The warped image has regions which were occluded, which are filled with information from the old image. As a result there are multiples of the same objects that generate a false error.


(a) Left image and right disparity map together with how the pixels are selected when warping the image

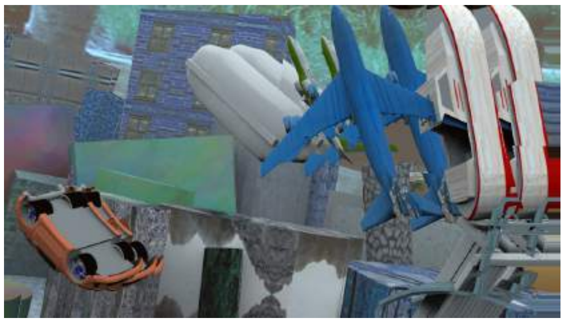

(b) Warped right image I'_r

Figure 3.13: The warped image I'_r has artifacts from the old image as there are occluded regions.

The similarity can also be measured using the structural similarity measure, SSIM, see Section 2.3.3.2. SSIM is used to calculates the similarity of the luminance, contrast and structure of the images for comparison. The resulting loss for the SSIM can be calculated as:

$$L_{\text{SSIM}} = \frac{1 - \text{SSIM}(I - I')}{2}. \tag{3.9}$$

The gradients, how much the intensity between a pixel and it neighbours differ, of the images can also be compared to see how similar the warped image and the original image are. The difference between the pixel-wise gradients in x and y-directions for the images can be written as the loss

$$L_{\text{grad}} = |\nabla I_x - \nabla I'_x| + |\nabla I_y - \nabla I'_y|, \tag{3.10}$$

where ∇ denotes the pixel-wise gradient. These three loss functions presented above are different ways of calculating the similarity between the reconstructed image and the original. They can be combined to a reconstruction loss:

$$L_{\text{rec}} = \frac{1}{N}\sum_{N}(\lambda_0 L_{\text{SSIM}} + \lambda_1 L_{\text{norm}} + \lambda_2 L_{\text{grad}}), \tag{3.11}$$

where λ specifies how the losses are weighted and N is the total number of pixels. The different loss functions are tested separately and then weighted together. Interesting to observe is that the SSIM loss, L_{SSIM}, and the gradient loss, L_{grad}, all work for networks that have been trained before, but for newly initialized networks the loss does not shrink at all, and all the disparity values converge to the mean value. This can be solved by initially using only the norm-error for training and after one epoch add the other losses, alternatively using a network trained for another dataset and then train it for new data using self-supervised learning, i.e. transfer learning.

When producing a disparity map, a common wanted feature of the predicted disparity map is to be locally smooth, i.e. the gradient between a pixel and it's neighbours is small. This can be obtained using a regularization loss:

$$L_{\text{reg}} = \frac{1}{N}\sum_{N}\Big(|\nabla_x^2 d|e^{-|\nabla_x^2 I|} + |\nabla_y^2 d|e^{-|\nabla_y^2 I|}\Big). \tag{3.12}$$

This regularization loss will, if used alone or weighted too high, make the network produce a disparity map with no difference between the gradients, i.e. same disparity throughout the whole map. An important feature of this network is to obtain a left-right consistency, i.e that the left and right disparity maps are related and consistent with each other. To obtain this the image warping technique can once again be used. An image can be warped twice, using both disparities, to reconstruct the original image, see Section 2.3.3. The loss for disparity consistency can be written as the absolute difference between an image and the double warped one. The consistency error can be written as:

$$L_{\text{con}} = |I - I''|, \tag{3.13}$$

where I'' is the image I warped two times.

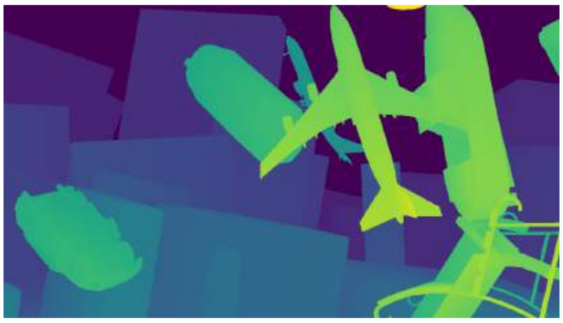
Also, to punish the high disparities in the occluded areas a maximum depth loss is added, and weighted very low. This loss function can be written as:

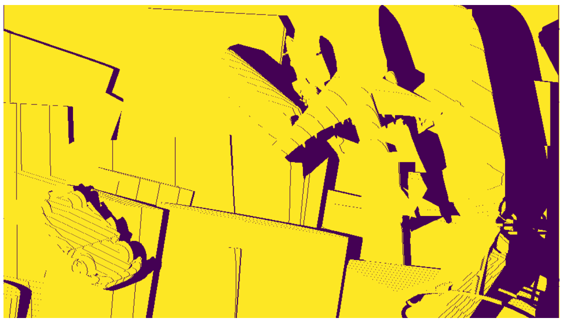
$$L_{\text{depth}} = \frac{1}{N}\sum_{N}|d|. \tag{3.14}$$

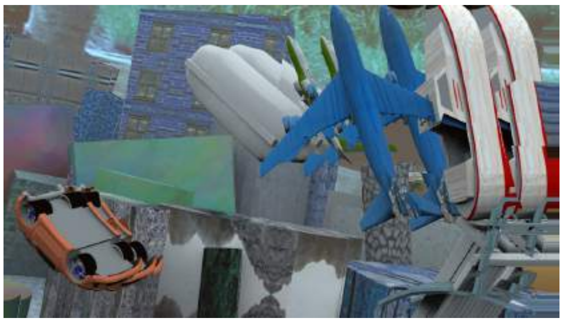
With a high weight, this loss will give a disparity map where all disparities are zero.

It is important to note that when bilinear sampling and warping is used on a dataset with occluded areas, information about what to fill the occluded areas with does not exist. The warped image will therefore have areas where the image information is not correct, even if the disparity used is the ground truth. Warped images containing large disparities will have larger regions with wrong information. In Figure 3.14 some objects are seen in two places, yielding a large reconstruction error, even though the disparity is correct.

A mask was created to ignore this error, by using the disparity map and finding occluded pixels. The mask is constructed from all the pixels in the warped image which the disparity does not point to. Pixels from the original target image are used to patch the warped image where there is a mask. The usage of the mask lowers the incorrect reconstruction loss. The network trained with the mask had an average outlier amount of 4.28 % while the unmasked had 4.60 %, showing that the masked network is better, considering both networks was trained on the same dataset.


(a) Ground truth right disparity map


(b) Generated mask from the disparity map, showing occluded regions in dark


(c) Warped right image with occluded regions

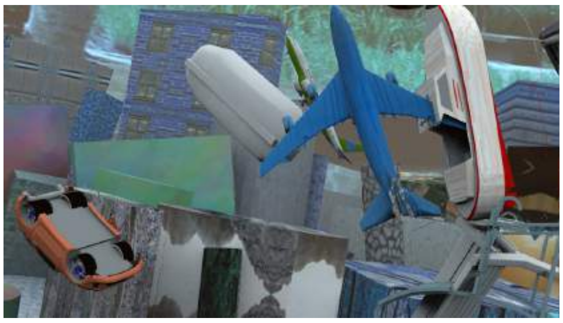

(d) Patched warped image, using the original right image where the mask is empty

Figure 3.14: The right image is warped to the left image using the ground truth left disparity and bilinear sampling. The warped image is patched with the original left image using the mask, where the right image has occluded regions.

The loss function combining all of the above presented loss functions can be written as:

$$L_{\text{tot}} = \omega_0(L^l_{\text{rec}} + L^r_{\text{rec}}) + \omega_1(L^l_{\text{reg}} + L^r_{\text{reg}}) + \omega_2(L^l_{\text{con}} + L^r_{\text{con}}) + \omega_3(L^l_{\text{depth}} + L^r_{\text{depth}}) \quad (3.15)$$

To decide in which manner the weights should be set, the approach was similar to the way the network was designed, i.e. training the network using different weights and then comparing the results iteratively. The main difference to this kind of iterative testing was that it is the way the loss is calculated that changes, and the resulting loss for the different approaches are not a valid comparison. Instead the number of outliers were compared for the networks trained with different weights, and weighted together with a visual estimation of how the network performs. Some differently weighed networks produce almost the same number of outliers, but different visual results. Some predictions gave disparity maps with the sky labeled as closer than the ground truth or darker areas as very far away, and these networks were ruled as unfit. In Table 3.2 some of the different evaluated weights can be seen with the average number of outliers for 200 images in the test data in Cityscapes. When the losses were designed and set initially, each separate loss was first given an initial weight and then combined and tuned to find a working network. The initial weights are found at Line 1 in Table 3.2. Not all tuning of the losses are presented in the table. The weights ω_1 and ω_3 were left unchanged, since altering them gave no preferable effect.

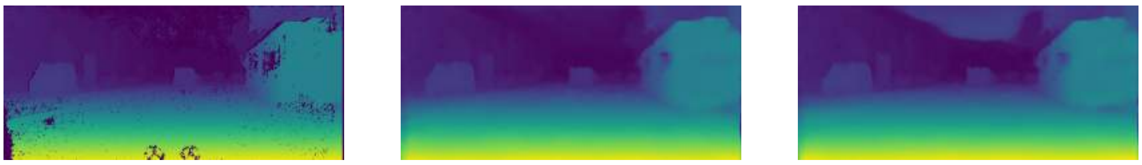
Table 3.2: The table presents some of the weights tried and their result after training 7 epochs on a tenth of the Cityscapes dataset. The outlier percentage is averaged over 200 samples of the test data. The best results are shown in bold.

	Weights							Outliers [%]		
Line	λ_0	λ_1	λ_2	ω_0	ω_1	ω_2	ω_3	$\tau_0 = 1$	$\tau_0 = 3$	$\tau_0 = 7$
1	0.5	0.8	0.5	1	0.001	0.5	0.001	39.8	6.9	1.6
2	0.5	0.8	0.5	0.5	0.001	0.5	0.001	32.9	5.7	1.7
3	0.3	0.8	0.5	1	0.001	0.5	0.001	29.4	5.7	1.7
4	0.5	0.8	0.5	1	0.001	1	0.001	33.0	5.9	1.7
5	0.5	0.2	0.5	0.5	0.001	0.5	0.001	38.4	6.8	1.7
6	0.5	0.6	0.25	0.5	0.001	0.5	0.001	27.9	5.3	1.5
7	0.5	0.8	0.25	0.5	0.001	0.5	0.001	**27.4**	5.3	1.6
8	0.5	0.8	0.35	0.5	0.001	0.5	0.001	27.8	**5.2**	**1.4**

An example of how an image with undesirable light and dark areas predicted with some weights can be seen in Figure 3.15. This kind of results were also taken into consideration when choosing the final weightings of λ and ω. In Figure 3.15c the sky has clearly got a higher disparity than what is right, ever though the weights used are line 6 in Table 3.2, comparable to line 7, producing the result in Figure 3.15b. Comparing line 7 and 8 in Table 3.2, the number of outliers produced are both small and the produced disparity maps looks much alike. The weights chosen as the best were therefore the weights on line 8, $\lambda = \{0.5, 0.8, 0.35\}$ and $\omega = \{0.5, 0.001, 0.5, 0.001\}$. These parameters were tuned for Cityscapes, and might not be optimal for other datasets.

3.3.3.2 Training

During this part of the network design process the normalization of the images were changed, so the intensity of the images were normalized between 0 and 1 for all color



(a) Ground truth left disparity

(b) Image which was seen as a good result

(c) Image which was seen as a bad result

Figure 3.15: An example of where two networks trained with different weights for the self-supervised loss produce relatively different output. The one producing a very light sky, relatively high disparity, is not seen as good when choosing the final weights.

channels. This was since we implemented the SSIM function to use pixel values in a range between 0 and 1. The optimizer used was Adam with the learning rate 0.001.

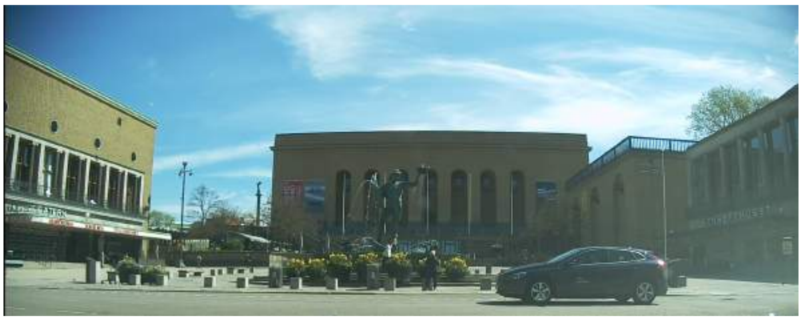
A network was trained supervised on the FT3D dataset for 16 hours before it was retrained for the new datasets. On Cityscapes the network was trained for 107 hours, but the loss was still declining. On Kitti the network was trained for 50 hours, but the validation loss was more or less constant after 24 hours. The network was also trained on the Carla dataset for the cameras with the shortest baseline for 118 hours, after which the validation loss was still shrinking.

3.4 Creating a new dataset

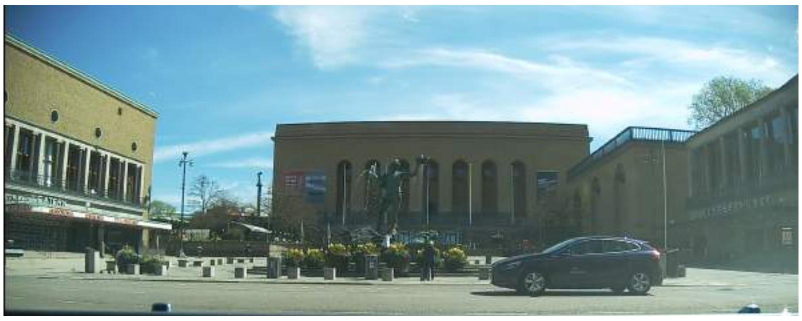
As a final step to test how applicable the network is in a real-life situation, the camera rig hardware was used to create a new dataset. By driving around in Gothenburg, stereo image pairs from an urban driving environment was captured. The data captured was called the Gbg dataset. The differences between this, the Cityscapes and the Kitti datasets are the camera parameters and different baseline between the cameras. A script as a proof of concept was implemented to capture the images. The script captured images with some delay between them, and when the rig was recording it was not exactly horizontal. Using OpenCV [14] the cameras were stereo calibrated and the images undistorted and rectified. However, the camera rig was not re-calibrated after installation in the vehicle, which resulted in the images not being aligned horizontally. Since the images were not horizontally aligned, the resulting disparity is both horizontal and vertical. Sample images from the Gbg dataset can be seen in Figure 3.16.

3.5 Summary

In this chapter the process of designing an ANN and tune its parameters has been discussed. The network design and building blocks were first chosen with regards to accuracy, runtime and memory requirements. The X-CNN was then chosen as the network design which could give a good enough accuracy in the shortest amount of time for full HD image pairs. The original design was tuned to find the best



(a) Gothenburg dataset, left image



(b) Gothenburg dataset, right image

Figure 3.16: A left and a right image from the Gbg dataset. The camera rig was not perfectly horizontal, and did not capture images at the same time with both cameras.

possible model. Multiple loss functions were implemented to enable unsupervised, or self-supervised, training of the model. The weights of the loss functions were optimized and a new dataset was captured using the camera rig.

The numerical and visual results from the final X-CNN with different training methods on different datasets are compared in the following chapter.

4
Results and discussion

In this chapter the results of the different network designs and training methods are presented first numerically and then visually. The results are followed by a discussion about the results obtained and how to interpret and improve them.

4.1 Numerical results

The evaluations of the networks' performance were done using the software Keras [47] with a TensorFlow [48] backend. The results from the networks depend on the ground truth in the datasets. Multiple models can be compared on the same dataset, but the errors for models are hard to compare between datasets. One problem is that the Kitti ground truth is very sparse, and another that Cityscapes sometimes have faulty annotations. The Carla dataset has a dense disparity map as ground truth. Results from the different supervised trained networks such as average runtime and average accuracy for 100 samples can be found in Table 4.1. The different errors calculated are presented in Section 3.3.

Table 4.1: Numerical results from the different supervised networks. Runtime is measured on a GeForce GTX 1080 Ti. The runtime and metrics are the average over 100 predictions.

	GC-lite	X-CNN	X-CNN
Training method	Supervised	Supervised	Supervised
Dataset	FT3D	FT3D	Carla
Training amount [epochs]	35	105	65
Training time [hours]	95	50	50
Input size [px]	768×512×3	960×540×3	1920×1080×3
Runtime [s]	0.9644	0.0524	0.2251
Outliers with $\tau_0 = 1$ [%]	14.411	25.930	17.707
Outliers with $\tau_0 = 3$ [%]	9.497	9.420	2.327
Outliers with $\tau_0 = 5$ [%]	7.777	5.537	1.371
Absolute relative error	0.184	0.100	1.423
Squared absolute relative error	6.367	2.774	2.859
Root mean square error	7.122	5.056	1.604
Mean absolute error, EPE	2.042	1.617	0.921

The GC-lite network was only trained on the FT3D data and results are therefore only presented for that specific dataset. The supervised X-CNN was able to handle larger input images and was therefore trained on different datasets to compare performance between different types of data. The X-CNN trained on the FT3D data has lower errors than the GC-lite network, as GC-lite was not optimized for better accuracy. The number of outliers for $\tau_0 = 3$ is almost the same, though the GC-lite has more perfect pixels, and therefore less outliers for $\tau_0 = 1$. The runtimes in the table are not easily compared since they depend on the number of pixels in the input, but the X-CNN is almost 20 times faster than the GC-lite network, even if it is used to predict on larger images.

The numerical results for the X-CNNs that were trained using self-supervision can be found in Table 4.2. The definition of the errors can be found in Section 3.3. One network, trained supervised for 17 epochs on a tenth of the FT3D data, was used as a start for all self-supervised training. All networks shown in the table were trained using self-supervised transfer learning from this network. As seen in the table, the self-supervised network performs relatively well on both the Kitti and the Cityscapes dataset, but has high errors for the Carla data set compared to the supervised network. This is a result of large errors from classifying the foreground as background, giving predictions containing areas with too low disparity. The visual outputs and errors can be seen in the figures in Section 4.2.

Table 4.2: Numerical results from the different self-supervised networks. The runtime is measured on a GeForce GTX 1080 Ti. The runtime and metrics are the average over 100 predictions, and the errors are calculated where a ground truth exists. The errors are only calculated for the left disparity as this is present in all the evaluated datasets.

	X-CNN	X-CNN	X-CNN
Training method	Self-supervised	Self-supervised	Self-supervised
Dataset	Kitti	Cityscapes	Carla
Training amount [epochs]	100	23	53
Training time [hours]	51	107	118
Input size [px]	1216×352×3	1984×896×3	1920×1080×3
Runtime [s]	0.0499	0.1538	0.2369
Outliers with $\tau_0 = 1$ [%]	28.212	18.234	20.295
Outliers with $\tau_0 = 3$ [%]	7.711	4.883	15.547
Outliers with $\tau_0 = 5$ [%]	4.605	2.968	12.039
Absolute relative error	0.056	0.082	2.483
Squared absolute relative error	0.507	1.250	33.410
Root mean square error	3.600	3.314	5.271
Mean absolute error, EPE	1.396	1.020	2.078

The runtimes are slightly higher for the self-supervised X-CNN than for the supervised version of the same network, since the self-supervised network has two outputs and the supervised only has one. However, the training time differs between them. The loss for the supervised X-CNN is lower than for the self-supervised X-CNN as the supervised network has been given the ground truth for the training data. The results for the self-supervised X-CNN trained on a dataset from real life shows promising results and accuracy. The accuracy on the Kitti dataset [39] for $\tau_0 = 3$ is at 7.711% which would result in a 82nd position on the Stereo benchmark in May 2018. The top performing network has 2.25 % outliers.

The runtime and resolution goals were further investigated. Different resolutions were tested on the TX2 and GeForce GTX 1080 Ti and the results can be found in Table 4.3.

Table 4.3: The average prediction time over 100 samples for the different GPUs. The resolution was altered to find an approximate frequency of 10 Hz.

GPU	Resolution [px]	Average runtime[s]	Frequency [Hz]
NVIDIA TX2	288×288×3	0.097	10.3
NVIDIA TX2	1920×1088×3	2.240	0.5
GeForce GTX 1080 Ti	1408×736×3	0.101	9.9
GeForce GTX 1080 Ti	1920×1088×3	0.237	4.2

4.2 Visual results

A visual evaluation of the networks were done to investigate where the network encounters problems. The results were used to better understand the network performance on different datasets and training types.

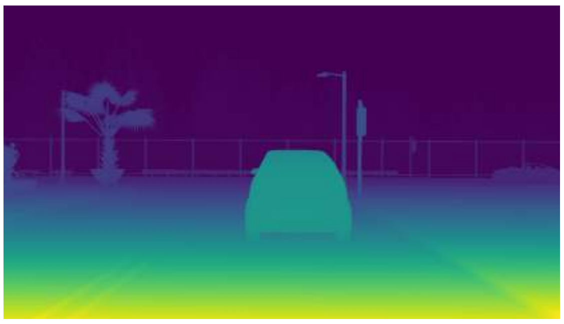
4.2.1 Representative scenes

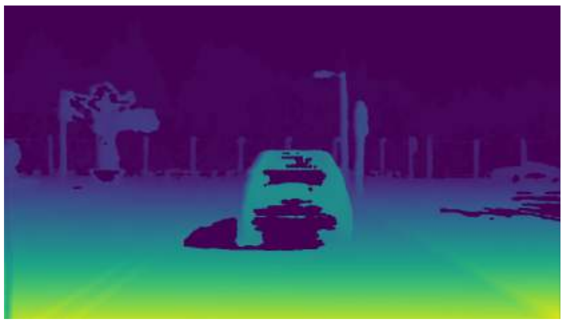
An example of how well the self-supervised X-CNN performs on Carla data can be found in Figure 4.1, Kitti in Figure 4.2, and on the Cityscapes data in Figure 4.3. It can be observed that the network performs poorly on the Carla data set. The visual results on the Cityscapes and Kitti data shows that the network has problem with the edges of objects and at the left edges of the images.

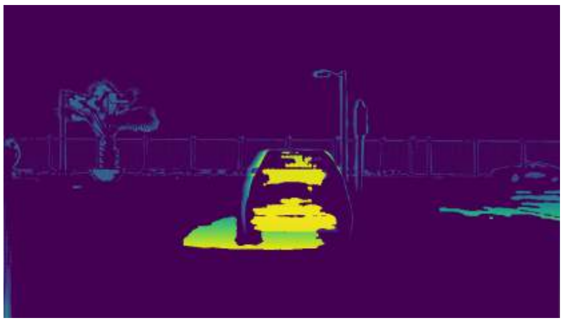
A comparison between the X-CNN trained using supervised training and the X-CNN using self-supervised training on the Carla data can be found in Figure 4.4. As seen from the resulting predictions the self-supervised X-CNN shows a smoother prediction without sharp edges, apparent in the lamp post and speed limit sign. This is since the self-supervised network receives no information about the occluded areas in the input images. For the Carla dataset the self-supervised X-CNN also shows no disparity for dark areas, such as the wall of the house and shadows in the bushes. This behaviour has not been seen to any larger extent in the netwroks trained on the real-life datasets Kitti and Cityscapes.

4.2.2 Highest errors

To better understand the network and how it works, the test data is used to find the images yielding the largest amount of outliers. This is done for the supervised X-CNN on the Carla dataset and for the self-supervised on the Cityscapes dataset.


(a) Ground truth left disparity


(b) Predicted left disparity


(c) Outlier map

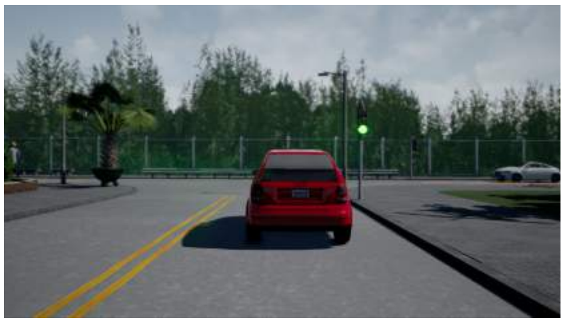
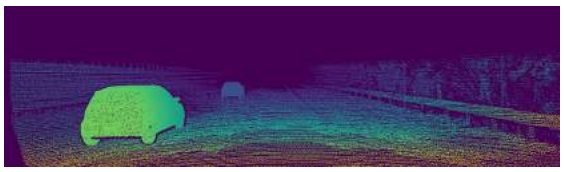
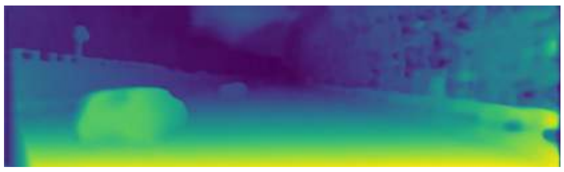
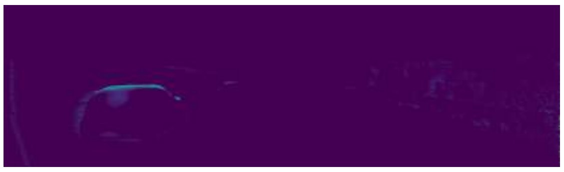

(d) Left input image

Figure 4.1: Result from the self-supervised X-CNN trained on the Carla data. For this prediction the number of outliers are 8.28 %. The maximum absolute error is 32 px.


(a) Ground truth left disparity


(b) Predicted left disparity


(c) Outlier map

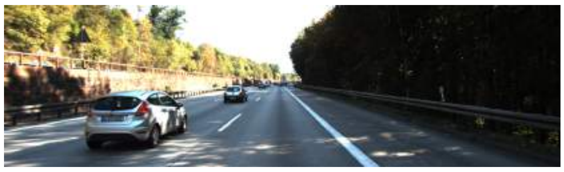
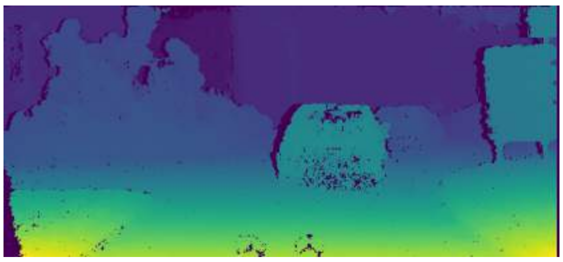
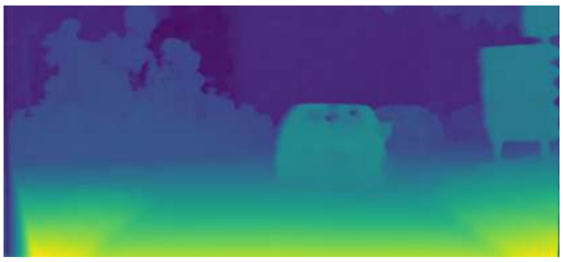

(d) Left input image

Figure 4.2: Result from the self-supervised X-CNN trained on the Kitti dataset. The outlier percentage for this prediction is 7.57 % and the max absolute error was 54 px.

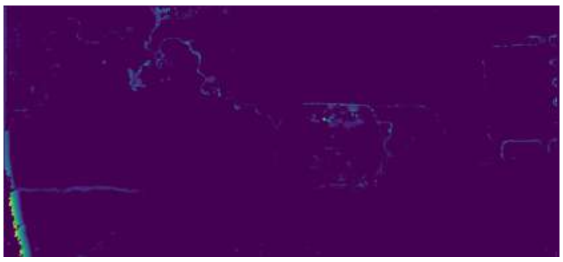
The prediction with the highest number of outliers for supervised training on Carla and self-supervised training on Cityscapes test data can be seen in Figure 4.5 and in Figure 4.6 respectively. Both image pairs contain objects very close to the cameras, which are not that common in the datasets.



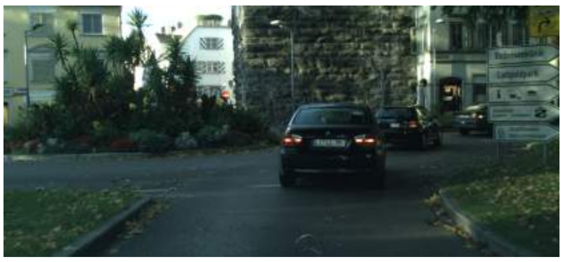
(a) Ground truth left disparity



(b) Predicted left disparity



(c) Outlier map



(d) Left input image

Figure 4.3: A prediction from the self-supervised X-CNN trained on the Cityscapes dataset. The number of outliers for this prediction is 3.38 %, and the maximum absolute error is 43 px.

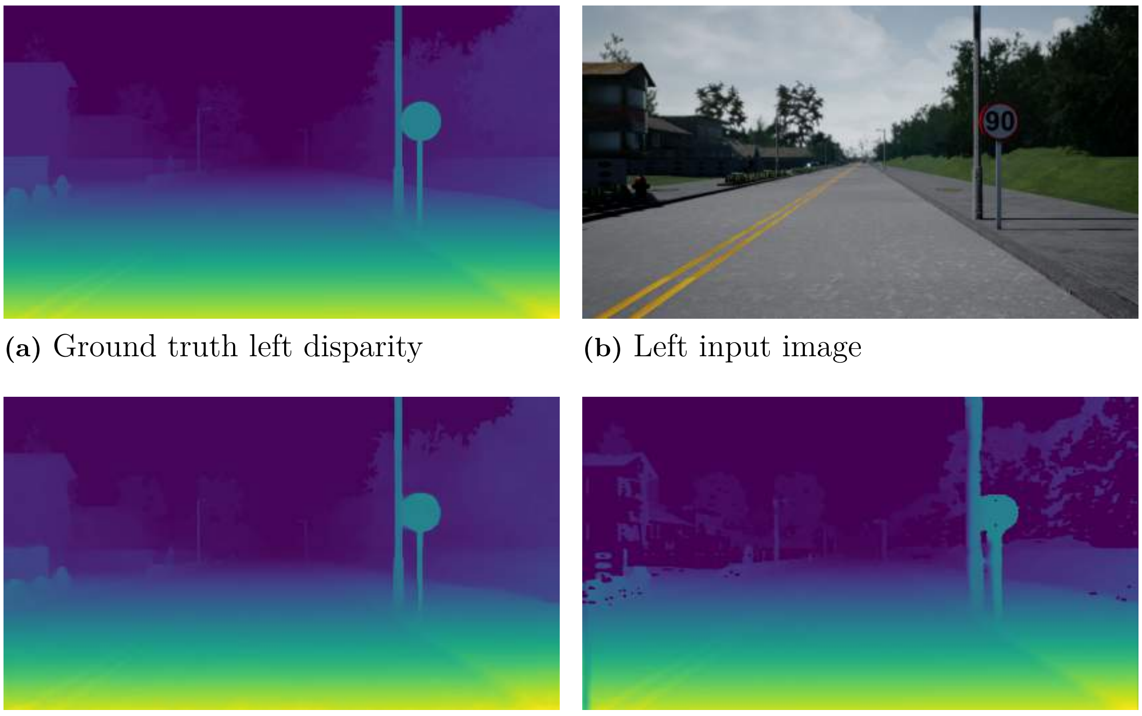
4.2.3 Visual evaluation

It can be concluded that the self-supervised version of the X-CNN works fairly well, especially on the real-life datasets. The results are better for the real-life datasets than for Carla which has problems with dark areas, as seen in Figure 4.1. Also, the problem with edges has increased using self-supervised learning. The lack of a clear shape for all the objects in the predicted disparity maps is a result of the occluded regions when warping the images.

By combining an image with the output disparity map, an image with depth for every pixel can be generated. A 3D image generated from the self-supervised X-CNN on an image pair from Cityscapes can be seen on the title page, as well as in Figure 4.7.

4.2.4 Gbg dataset

The model that was trained on Cityscapes dataset was retrained on the Gbg dataset. The dataset was created as a proof of concept, to show how the network and camera rig could be used together. The dataset was not captured with the cameras exactly horizontal, which resulted in a vertical disparity between the images. The images were captured in series, which created a short time delay between the two images. In Figure 4.8, an image pair, output disparity map and the warped image, I_l', can be seen. It is difficult to calculate the accuracy of the model, as no ground truth exists, but the warped image compared to the original left image gives a visual indication of the accuracy. A numerical accuracy measure might be constructed between those images, but the results were not evaluated further. Figure 4.8 shows that the network



(a) Ground truth left disparity **(b)** Left input image

(c) Prediction from supervised network **(d)** Self-supervised network prediction

Figure 4.4: A comparison between the predictions produced by the networks trained supervised and self-supervised. The self-supervised network produces smoother edges, seen in the lamp post. For the Carla data the self-supervised network experiences problems with dark colors, such as shadows.

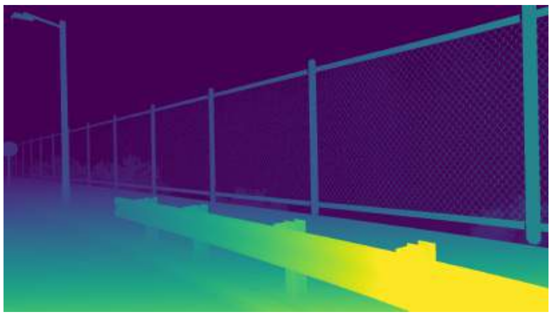
is able to generate a disparity map from pictures taken by our camera. The warped loss assumes rectified images taken horizontally, and the vertical disparity makes the loss faulty. To get a more correct disparity map, the images needs to be horizontally rectified, the camera calibration more accurate, the dataset larger and the images needs to be captured synchronously.

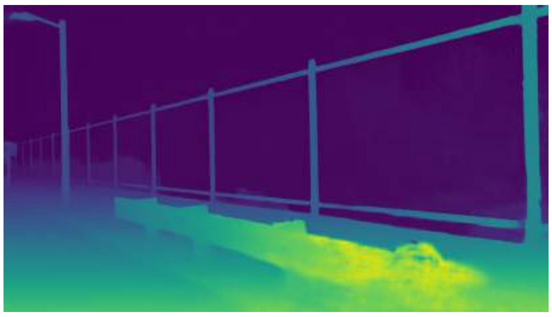
4.3 Discussion

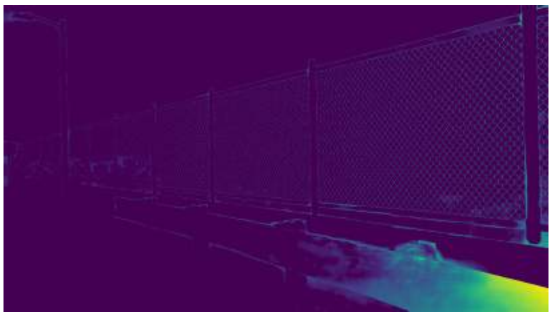
In this section, the key aspects of the thesis such as the network design, computational power, new datasets and future work are discussed.

4.3.1 Network performance

The results presented show that the final network structure, the X-CNN, is not fast enough to be able to predict a full HD disparity map at a frequency of 10 Hz on the TX2. A prediction using the self-supervised X-CNN on the TX2 takes 2.24 s. However, the network design is slim enough for the TX2 to be able to predict full HD images without running out of memory. The accuracy of the network is not as high as the best performing network on the Kitti benchmark scoreboard, but still


(a) Ground truth left disparity


(b) Predicted disparity


(c) Outlier heat map

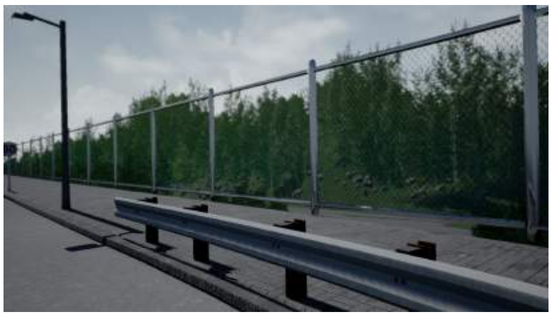
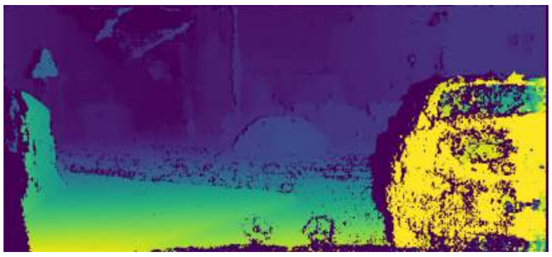
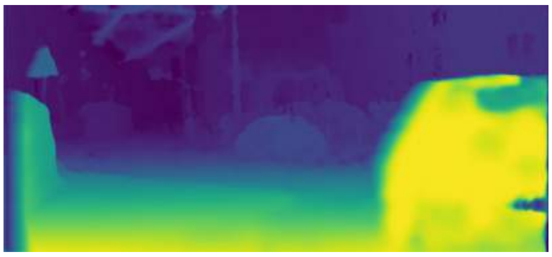
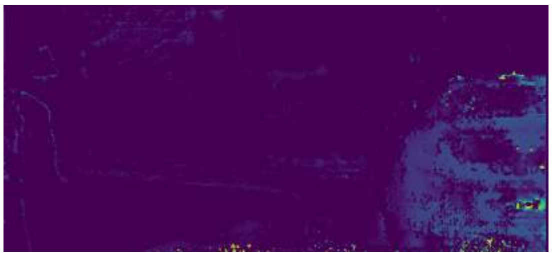

(d) Left input image

Figure 4.5: The data producing the most outliers from the Carla test data for the supervised X-CNN. The outlier percentage is 13.4 % and the maximum absolute error is 61 px.


(a) Ground truth left disparity


(b) Predicted disparity


(c) Outlier heat map

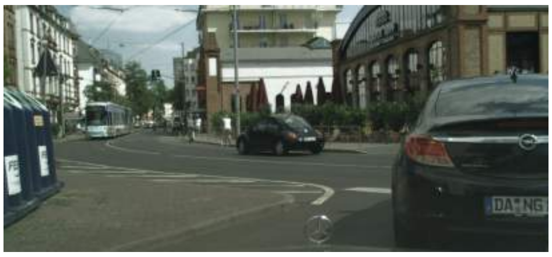
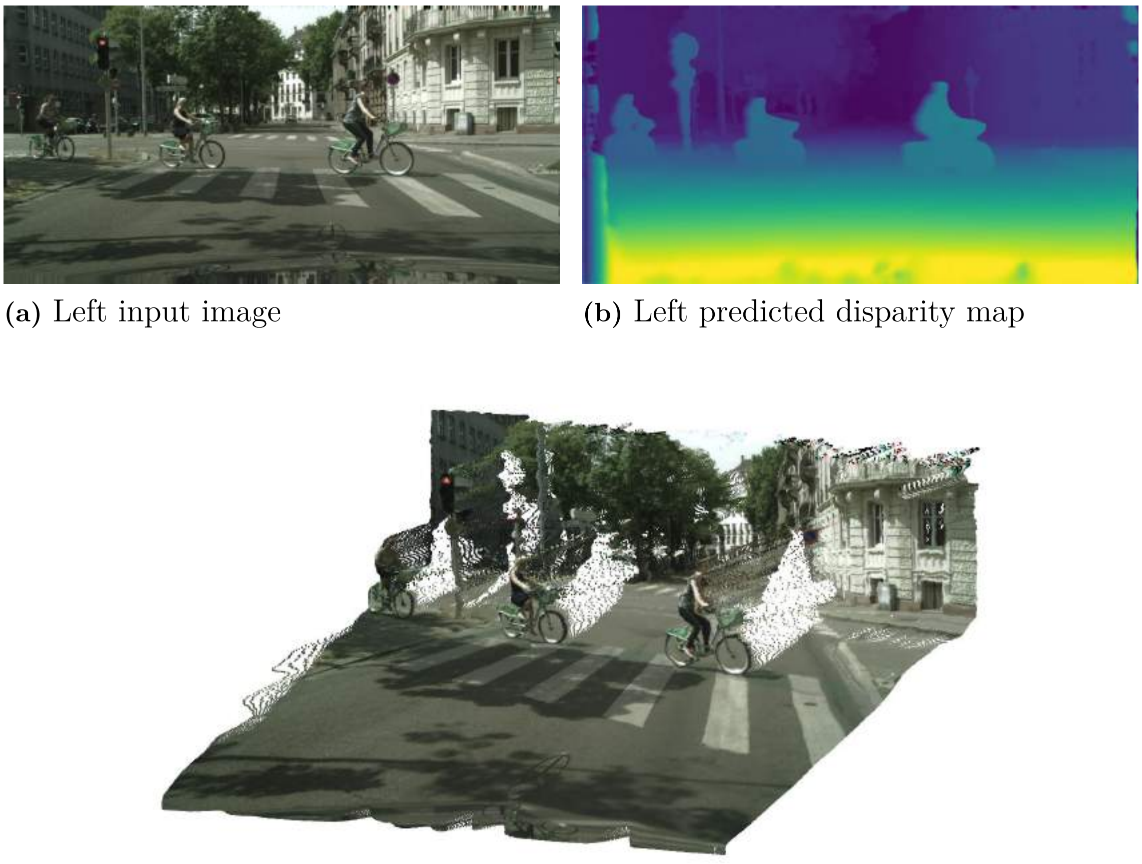

(d) Left input image

Figure 4.6: The data producing the most outliers from the Cityscapes test data for the self-supervised X-CNN. The maximum absolute error for this prediction is 101 px and the outlier percentage is 19.4 %



(a) Left input image (b) Left predicted disparity map

(c) Calculated 3D image by combining the image and the disparity map

Figure 4.7: A 3D image calculated with the self-supervised X-CNN using an image pair from Cityscapes. The white areas are occluded regions.

outperforms some of the algorithms. The resolution needed to fulfill the runtime goal for a TX2, was much lower than full HD. A trade-off between runtime and resolution can be done to find a balance.

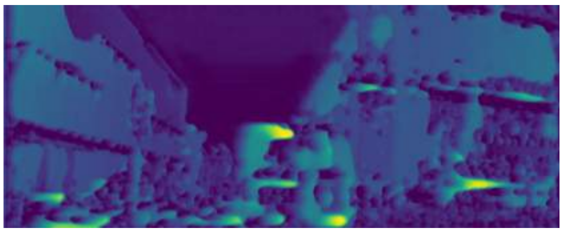
The supervised X-CNN trained on the FT3D dataset has an outlier percentage of 9.4 % for $\tau_0 = 3$, and an EPE of 1.6 for one prediction, taking 0.05 s. This can be compared to [50] which presents a network producing an outlier percentage of 6.2 % and an EPE of 1.3, but with a runtime of almost 0.5 s which is ten times as long runtime as X-CNN.

The self-supervised X-CNN's outlier percentage is 7.7 % for the Kitti benchmark with a runtime of 0.05 s. This can be compared to the self-supervised network SsSMnet [30], which is also trained on Kitti. The SsSMnet has a 3.4 % error, but a runtime of 0.8 s on a Tesla P100 GPU which is a stronger GPU than the one we have used. The X-CNN has not been submitted to Kitti for exact evaluation, which means that the networks are not evaluated on the same data.

The numeric results show that the X-CNN is a network which gives rather good results at a high frequency. The visual results shows that the predicted disparity maps mostly have minor errors, such as the edges of different objects. We have also

(a) Left image

(b) Right image


(c) Calculated disparity map

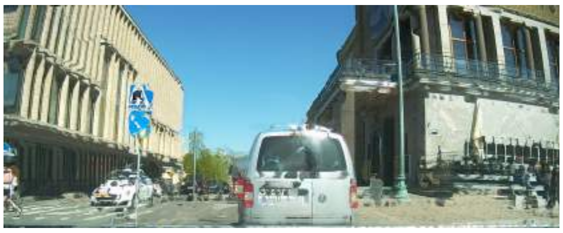

(d) Warped left image

Figure 4.8: A real life application of the network. As the left and right images are not captured horizontally, close objects in the warped image are duplicated and generates an error.

showed that the self-supervised network can be transfer trained to work in different applications.

4.3.2 Self-supervision

A self-supervised network has generally lower accuracy than the supervised equivalent. The largest advantage with self-supervised training is that no ground truth is needed. By eliminating the need for ground truth data, the number of applications in which the system could be used is increased drastically. The camera rig could be used to collect data manually on which it could be trained, the network could then be used in that environment. Datasets such as Kitti has a combination of data with and without ground truth. On such a dataset a combination of supervised and self-supervised training might improve the accuracy.

4.3.3 Robustness and adaptability

A network trained on a dataset that is not biased, such as the FT3D dataset, can be trained using transfer learning to learn new environments. The adaptability is needed to make the camera rig useful in different environments. By using the LiDAR point distance sensor on the camera rig, a single point ground truth can be obtained when gathering new data. This data point could be used in a loss function to make the self-supervised training slightly supervised. The LiDAR point could also be used to estimate the live accuracy of the disparity estimation, to verify the output of the network.

4.3.4 Computing power

As stated in the evaluations of the network designs in Section 3, the goal of predicting full HD disparity maps on the TX2 at the frequency of 10 Hz was decided to be unachievable. However, for the camera rig to predict in that frequency would still be possible with a better GPU or some other kind of fast computing device. Another way to tackle the problems of the computational demand would be to reduce the size of the input to the network. In many applications there might be no need for full HD resolution outputs. If the hardware onboard the camera rig had more computational power the network could be trained online, and the produced disparity estimate be improved during usage of the rig.

4.3.5 Trade-offs

When using ANNs, as well as when using conventional algorithms for disparity estimation, there exists a trade-off between accuracy, memory usage and computational time. The introductions of new neural network designs and techniques have allowed for higher accuracies, but to the cost of more computational power. The need of good accuracy for high resolution images at real-time is a difficult balance. This trade-off exists in almost all computer vision tasks. When a network has been trained for a dataset, the network can be pruned and retrained with the same data to optimize the runtime. Pruning could lower the runtime, but also limit the network adaptability to changes in the dataset and environment.

4.3.6 Applying to real life

When using the camera rig to generate new datasets, a number of problems occurred. If the rig is mounted on a vehicle, it might not be stable around the roll axis. If the cameras are not horizontally aligned, and the calibration does not compensate for that, the feature matching will have to be done both horizontally and vertically, and objects in the warped image may be duplicated. The functions for the bilinear sampling and for calculating the mask used in the self-supervised loss functions are not written for handling vertical disparity. All data used for network training has been rectified, and the images have to be captured synchronously. If the images are not captured at the exact same time, objects might move. This would lead to a disparity between the images not correlated with depth. When the Gbg dataset was captured the camera rig was not exactly horizontal, which it had been for calibration, and the images were not captured at the same time. The network still produces a disparity map, though not very accurate.

4.3.7 Disparity range

This thesis focused on using one stereo camera pair, where the design parameter of a maximum depth of 200 has been used multiple times, for example when denormalizing the disparity in the self-supervised X-CNN. This is a design parameter suitable for camera pairs with a short baseline and for predicting disparities corresponding to a minimum depth of a little less than a meter. However, for other camera pairs on the camera rig the design parameter d_{max} has to be changed.

4.3.8 Multiple cameras in stereo

The X-CNN uses two input images and outputs two disparity maps. How to utilize the full potential of the camera rig, using all four cameras, is something that needs further investigation. For example a solution could be to alter the network to support multiple stereo pairs, leading to longer runtimes, or to feed the network with an extra number as input corresponding to the baseline. In addition, if the network produces multiple disparity maps for different camera pairs or if multiple networks are used, the disparity maps need to be weighted together to produce a more accurate disparity for larger ranges.

5 Conclusion

We have introduced a novel artificial neural network, a cross-connected CNNs, called X-CNN, which can handle two high resolution images as input and calculate the corresponding disparity maps. The network is a Siamese encoder-decoder structured CNN that is cross connected after each layer. The network's loss functions depend only on the input images and the predicted disparities. The loss function is dependent on warping one of the images to the other by using one of the predicted disparities. By presenting a mask to filter away the faulty errors from occluded areas in the images, a self-supervised network could be designed. The network showed a high accuracy for supervised learning at a 4 Hz rate for a full HD stereo image pair on a GeForce GTX 1080 Ti. Further, the network could be transfer trained to new datasets with different camera parameters, and was for example trained self-supervised on the Cityscapes dataset where it produced 4.89 % outliers on the test data.

The purpose of this project was to design a self-supervised ANN which could adapt to new situations. However, the goal of predicting full HD disparity maps at a frequency of 10 Hz, has not been fulfilled. Connected to this are some future work that could improve the results of the Cross-CNN. The speed of the network can be improved by applying pruning, but pruning would also limit the networks adaptability. There are environments that the camera rig could be used in, where no datasets today exist. If new datasets were created, the network could be trained for these to verify its adaptability. How to take advantage of the camera rig's four cameras to refine the estimation should be investigated further. In a real-world application the camera rig could be affected by movement in roll, so that the cameras are not horizontally aligned. We have showed that this is a problem for the network today, and that a need for future improvement exists.

Bibliography

[1] "Volvo first in the world with self-driving truck in underground mine," http://www.volvogroup.com/en-en/news/2016/sep/news-2297091.html, 2016, [Online; accessed: Feb. 9, 2018].

[2] "Volvo pioneers autonomous, self-driving refuse truck in the urban environment," http://www.volvogroup.com/en-en/news/2017/may/news-2561936.html, 2017, [Online; accessed: Feb. 9, 2018].

[3] J. C. Knight, "Safety critical systems: challenges and directions," in *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*, May 2002, pp. 547–550.

[4] R. Andersson and O. Noresson, "Utilization of quadnocular stereo vision for simultaneous localization and mapping in autonomous vehicles," Master's thesis, Chalmers tekniska högskola, 2015.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[6] A. Kendall, H. Martirosyan, S. Dasgupta, P. Henry, R. Kennedy, A. Bachrach, and A. Bry, "End-to-end learning of geometry and context for deep stereo regression," *CoRR*, vol. abs/1703.04309, 2017. [Online]. Available: http://arxiv.org/abs/1703.04309

[7] C. Zhou, H. Zhang, X. Shen, and J. Jia, "Unsupervised learning of stereo matching," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1567–1575.

[8] C. Godard, O. Mac Aodha, and G. J. Brostow, "Unsupervised monocular depth estimation with left-right consistency," *arXiv preprint arXiv:1609.03677*, 2016.

[9] Y. Movshovitz-Attias, T. Kanade, and Y. Sheikh, "How useful is photo-realistic rendering for visual learning?" *CoRR*, vol. abs/1603.08152, 2016. [Online]. Available: http://arxiv.org/abs/1603.08152

[10] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library.* " O'Reilly Media, Inc.", 2008, pp. 371-374.

[11] ——, *Learning OpenCV: Computer vision with the OpenCV library.* " O'Reilly Media, Inc.", 2008, pp. 375-377.

[12] R. B. Fisher and E. C. (e-book collection), *Dictionary of computer vision and image processing*, 2nd ed. Chichester, West Sussex: John Wiley & Sons Inc, 2014;, p. 98.

[13] R. Klette, B. (e-book collection), and S. (e-book collection), *Concise computer vision: an introduction into theory and algorithms*, 2014th ed. London: Springer, 2014, pp. 227-233.

[14] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[15] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library.* " O'Reilly Media, Inc.", 2008, pp. 378-395.

[16] ——, *Learning OpenCV: Computer vision with the OpenCV library.* " O'Reilly Media, Inc.", 2008, pp. 427-431.

[17] R. Klette, B. (e-book collection), and S. (e-book collection), *Concise computer vision: an introduction into theory and algorithms*, 2014th ed. London: Springer, 2014, pp. 287, 331.

[18] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004.

[19] M. Sonka, V. Hlavac, and R. Boyle, *Image processing, analysis, and machine vision.* Cengage Learning, 2014, pp. 385-388.

[20] J. Ulén, "Higher-order regularization in computer vision," Ph.D. dissertation, Lund University, 2014.

[21] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International journal of computer vision*, vol. 47, no. 1-3, pp. 7–42, 2002.

[22] N. Buduma and N. Locascio, *Fundamentals of Deep Learning: Designing Next-generation Machine Intelligence Algorithms.* " O'Reilly Media, Inc.", 2017, pp. 21-23.

[23] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016, http://www.deeplearningbook.org, p. 174.

[24] ——, *Deep Learning.* MIT Press, 2016, http://www.deeplearningbook.org, pp. 330-339.

[25] H. Habibi Aghdam, E. Jahani Heravi, S. O. service), and S. (e-book collection), *Guide to Convolutional Neural Networks: A Practical Application to Traffic-Sign Detection and Classification.* Cham: Springer International Publishing, 2017, pp. 86-90.

[26] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller, "Striving for simplicity: The all convolutional net," *CoRR*, vol. abs/1412.6806, 2014. [Online]. Available: http://arxiv.org/abs/1412.6806

[27] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, "Deconvolutional networks," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, June 2010, pp. 2528–2535.

[28] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016, http://www.deeplearningbook.org, pp. 105-109.

[29] ——, *Deep Learning.* MIT Press, 2016, http://www.deeplearningbook.org, pp. 302-306.

[30] Y. Zhong, Y. Dai, and H. Li, "Self-supervised learning for stereo matching with self-improving ability," *CoRR*, vol. abs/1709.00930, 2017. [Online]. Available: http://arxiv.org/abs/1709.00930

[31] T. Semwal, G. Mathur, P. Yenigalla, and S. B. Nair, “A practitioners’ guide to transfer learning for text classification using convolutional neural networks,” *CoRR*, vol. abs/1801.06480, 2018. [Online]. Available: http://arxiv.org/abs/1801.06480

[32] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding,” *CoRR*, vol. abs/1510.00149, 2015. [Online]. Available: http://arxiv.org/abs/1510.00149

[33] W. Luo, A. G. Schwing, and R. Urtasun, “Efficient deep learning for stereo matching,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 5695–5703.

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

[35] C. A. Glasbey and K. V. Mardia, “A review of image-warping methods,” *Journal of Applied Statistics*, vol. 25, no. 2, pp. 155–171, 1998.

[36] K. T. Gribbon and D. G. Bailey, “A novel approach to real-time bilinear interpolation.” IEEE, 2004, pp. 126–131.

[37] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, “Spatial transformer networks,” *CoRR*, vol. abs/1506.02025, 2015. [Online]. Available: http://arxiv.org/abs/1506.02025

[38] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, “Image quality assessment: from error visibility to structural similarity,” *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, April 2004.

[39] M. Menze and A. Geiger, “Object scene flow for autonomous vehicles,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[40] L. Caltagirone, “Deep learning applications for autonomous driving,” Ph.D. dissertation, 2018.

[41] “Nvidia jetson tx2,” https://developer.nvidia.com/embedded/buy/jetson-tx2-devkit, 2018, [Online; accessed: Jan. 4, 2018].

[42] “Li-imx185-mipi-cs,” https://leopardimaging.com/product/li-imx185-mipi-cs/, 2018, [Online; accessed: Mar. 19, 2018].

[43] “Sf30-c high speed laser rangefinder - 100 m,” https://www.parallax.com/product/28058, 2018, [Online; accessed: Apr. 26, 2018].

[44] N.Mayer, E.Ilg, P.Häusser, P.Fischer, D.Cremers, A.Dosovitskiy, and T.Brox, “A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation,” in *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, arXiv:1512.02134. [Online]. Available: http://lmb.informatik.uni-freiburg.de/Publications/2016/MIFDB16

[45] 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.

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

[47] F. Chollet *et al.*, “Keras,” https://keras.io, 2015.

[48] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. 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 systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[49] Z. Yan and X. Xiang, "Scene flow estimation: A survey," *CoRR*, vol. abs/1612.02590, 2016. [Online]. Available: http://arxiv.org/abs/1612.02590

[50] J. Pang, W. Sun, J. S. J. Ren, C. Yang, and Q. Yan, "Cascade residual learning: A two-stage convolutional neural network for stereo matching," *CoRR*, vol. abs/1708.09204, 2017. [Online]. Available: http://arxiv.org/abs/1708.09204

A
Appendix 1

The design of X-CNN can be found in Table A.1.

Table A.1: A description of the layers in the X-CNN network.

	Layer Description	Output Dim.
	Input images	W,H,C
	Every layer is duplicated for the left and right network which shares weights. Every layer is also added to the same layer in the other network with a scalar weight	
	Convolutions	
1	7 × 7 conv, 20 features, stride 2	$W/2, H/2, 20$
2	5 × 5 conv, 20 features	$W/2, H/2, 20$
3	5 × 5 conv, 20 features	$W/2, H/2, 20$
4-6	Repeat 1-3 with filter size 5, 3, 3	$W/4, H/4, 20$
7-9	Repeat 4-6 with 40 features	$W/8, H/8, 40$
10-12	Repeat 7-9 with filter size 3, 3, 3	$W/16, H/16, 40$
13-15	Repeat 10-12 with 80 features	$W/32, H/32, 80$
	Deconvolutions	
16	3 × 3 deconv, 80 features	$W/32, H/32, 80$
17	3 × 3 deconv, 80 features, stride 2	$W/16, H/16, 80$
	add layer 17 and 12 (residual connection)	
18	3 × 3 deconv, 40 features	$W/16, H/16, 40$
19-20	Repeat 17-18 with 40 features	
	add layer 19 and 9 (residual connection)	$W/8, H/8, 40$
21-22	Repeat 17-18 with filter size 5, 3 and 20 features	$W/4, H/4, 20$
23-24	Repeat 21-22	$W/2, H/2, 20$
25	5 × 5 deconv, 20 features, stride 2	$W, H, 20$
	Output layer	
26	5 × 5 conv, 1 feature	$W, H, 1$