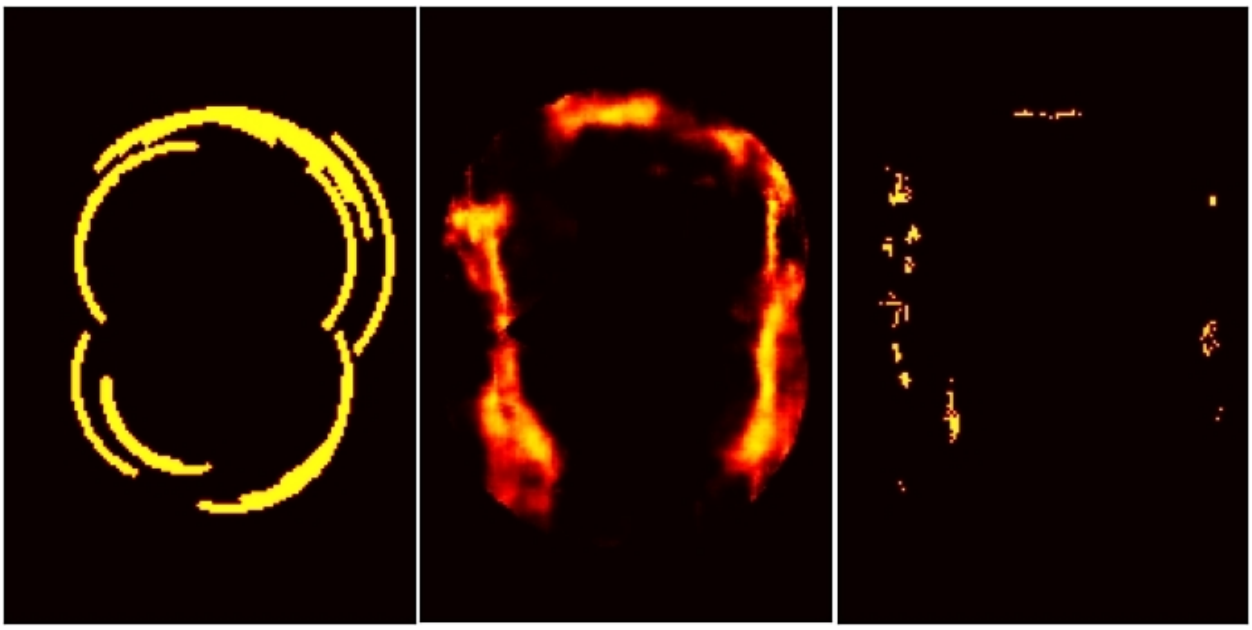
CHALMERS
UNIVERSITY OF TECHNOLOGY



Estimation of Lidar Point Clouds Based on Ultrasonic Sensors Using Deep Learning

Master's thesis in Complex adaptive systems, Engineering mathematics and computational science

Carl Hjalmarsson and Jesper Jäghagen

DEPARTMENT OF MATHEMATICAL SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Master's thesis 2023

Estimation of Lidar Point Clouds Based on Ultrasonic Sensors Using Deep Learning

Carl Hjalmarsson

Jesper Jäghagen

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Chalmers University of Technology
Gothenburg, Sweden 2023

Estimation of Lidar Point Clouds Based on Ultrasonic Sensors Using Deep Learning
Carl Hjalmarsson and Jesper Jäghagen

Supervisor: Andreas Abramsson, Volvo Cars
Examiner: Adam Andersson, Department of Mathematical Sciences

Master's Thesis 2023
Department of Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Three maps of a vehicle environment or occupancy grids created by (Left) ultrasonic sensors, (Middle) a trained neural network using the left image as input and (Right) high precision Lidar sensors. The Lidar map is used as ground truth.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

Estimation of Lidar Point Clouds Based on Ultrasonic Sensors Using Deep Learning
Carl Hjalmarsson and Jesper Jäghagen
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

Safety is a key point of research in the automotive industry. Car companies dedicate a great amount of time and resources to make their cars safer by developing sensory systems like Low Speed Perception (LSP). In this thesis, we have explored the possibility to enhance the contribution of ultrasonic sensors to LSP by leveraging deep learning to mimic data from the more expensive Lidar sensors.

To do this we draw inspiration from three different deep learning approaches: image denoising, image segmentation and image-to-image translation, resulting in five different models: $\text{DAE}^{(\text{bin})}$, $\text{DAE}^{(\text{mse})}$, $\text{UNET}^{(\text{bin})}$, $\text{UNET}^{(\text{ce})}$ and an I2I cGAN. We train these models on three datasets, each containing paired ultrasonic and Lidar representations of a two-dimensional environment around the car. In order to measure the performance of these models, we develop an evaluation framework where we assess the ability of the models to map ultrasonic object detections to corresponding Lidar detections.

We find that the performance of all networks is highly dependent on the data representation. When using a basic representation, consisting only of point detections and free-space, all models fail to improve upon the baseline ultrasonic sensor score. When using a sparse representation consisting of detection arcs, however, $\text{UNET}^{(\text{bin})}$ succeeds in outperforming the baseline and mimicking the more accurate Lidar representation (see cover). Finally, when adding unknown areas of the environment to the sparse representation, both $\text{UNET}^{(\text{ce})}$ and the cGAN manage to outperform the baseline in most aspects, and we see a convergence towards the more realistic Lidar representation.

The results show that there is indeed a possibility to enhance ultrasonic sensor perception using deep learning and Lidar reference data, and while there is still much room for improvement, we have shown that there is potential in further research on this task.

Keywords: Low Speed Perception, Ultrasonic Sensor Perception, Deep Learning, Autoencoders, U-Net, Conditional Generative Adversarial Networks.

Acknowledgements

This Master's thesis project is possible thanks to the access of vehicle data and computational resources provided by Volvo Cars. We would also like to thank our supervisor Andreas Abramsson for the support and materials provided during the project. Further more, a big thanks to the low speed perception team, Team Dolphins, and artificial intelligence research team, Team Sunnyvale, for insights in the sensor fusion and deep learning fields, respectively. Finally, a salutation to our opponents Anton Rosenberg and Viktor Wiklund for valuable feedback.

Carl Hjalmarsson and Jesper Jäghagen, Gothenburg, June 2023

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

ADAS	Advanced Driver Assistance Systems
AE	Autoencoder
BCE	Binary Cross Entropy
CE	Cross Entropy
cGAN	Conditional Generative Adversarial Network
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DAE	Denoising Autoencoder
Disco-GAN	Discover Cross-Domain Generative Adversarial Network
FFNN	Feedforward Neuaral Network
FN	False Negative
FP	False Positive
GAN	Generative Adversarial Network
GT	Ground Truth
GPU	Graphics Processing Unit
HDF5	Hierarchical Data Format 5 (File)
I2I	Image to Image
LSP	Low Speed Perception
MAE	Mean Absolute Error
MSE	Mean Squared Error
NN	Neural Network
OG	Occupancy Grid
PCAP	Package Capture (File)
RMSE	Root Mean Squared Error
TN	True Negative
TP	True Positive
UNIX	Uniplexed Information Computing System
USS	Ultrasonic Sensor
VCC	Volvo Cars
WGAN	Wasserstein Generative Adversarial Network

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

Indices

i,j,k,r,p	Indices for neurons in a hidden layer
t	Index for time step

Sets

$\mathcal{D}_x$	Set of input data
$\mathcal{D}_y$	Set of target data
$\mathcal{D}_x^{(point)}$	Point input dataset - USS grids containing only occupied and free-space cells
$\mathcal{D}_x^{(sparse\ arc)}$	Sparse arc input dataset - USS grids containing arcs of occupied cells and free-space cells
$\mathcal{D}_x^{(dense\ arc)}$	Dense arc input dataset - USS grids containing arcs of occupied cells, free-space cells and unknown cells
$\mathcal{D}_y^{(sparse)}$	Sparse ground truth dataset - Lidar grids containing occupied and free-space cells
$\mathcal{D}_y^{(dense)}$	Dense ground truth dataset - Lidar grids containing occupied, free-space and unknown cells
$\mathcal{D}^{(basic)}$	Basic dataset - Paired point input and sparse ground truth dataset
$\mathcal{D}^{(sparse)}$	Sparse dataset - Paired sparse arc input and sparse ground truth dataset
$\mathcal{D}^{(dense)}$	Dense dataset - Paired dense arc input and dense groundtruth dataset

Parameters

α	Learning rate
λ	MAE loss weighting parameter
m_b	Batch size
n	Number of patterns in a dataset

Variables

$\boldsymbol{x}$	Input pattern or USS frame
$\boldsymbol{y}$	Ground truth pattern or Lidar frame
$\hat{\boldsymbol{y}}$	Predicted Lidar frame
$\boldsymbol{W}$	Neural network weights
$\boldsymbol{\Theta}$	Neural network biases
r	Radial distance from sensor location to detection
d	Distance
$\boldsymbol{\phi}$	Sensor resolution angle
$\boldsymbol{\theta_i}$	Sensor direction angle

Functions

$\mathcal{L}$	Loss function
G	Generator mapping
D	Discriminator mapping
p_d	Data distribution
$p_{\boldsymbol{z}}$	Prior noise distribution
p_g	Generator network output distribution

Contents

List of Figures

List of Tables

1

Introduction

In the realm of automotive engineering, safety is a paramount concern. In the last few decades, new technology has ushered this realm into a new epoch. As a result, the industry has witnessed remarkable advancements in ensuring the well-being of vehicle occupants and minimizing the risks associated with accidents. This is reflected in car companies such as Volvo Cars, which now are devoted to a zero collisions policy [29].

Two key pillars that underpin automotive safety are active safety and passive safety. Active safety refers to the proactive measures and technologies implemented to prevent accidents or mitigate their severity in real-time. This includes features such as Advanced Driver Assistance Systems (ADAS) and collision avoidance systems. In contrast, passive safety measures are designed to safeguard occupants during a collision or impact. This includes the structural integrity of the vehicle, airbags, seat belts, and crumple zones.

With regards to ADAS and collision avoidance systems, new sensor technology and digitalization have made it possible to gather a large amount of information about environment around the car. This has enabled the development of functions such as the auto-reverse brake, which uses radar or ultrasonic sensors to stop the car when an object is too close.

In the Volvo Car Corporation (VCC), the auto-reverse brake constitutes a single function among a multitude of capabilities that leverage the continuously evolving Low Speed Perception (LSP) system. This system employs a fusion of sensor data to construct a comprehensive model of the environment during low-speed scenarios. The sensors responsible for collecting this data can be categorized into two types: production sensors and reference sensors. Production sensors, intended for consumer vehicles, are generally smaller and more cost-effective. In contrast, reference sensors are exclusively employed in test vehicles for the purposes of development and validation. An example of a high-precision reference sensor is the **Lidar** (Light Detection and Ranging) sensor, which generates a three-dimensional point cloud depicting the nearby environment [30].

One aspect of LSP is **UltraSonic Sensor** (USS) perception, where a map of the environment surrounding the vehicle is created by firing an array of ultrasonic sensors

in different sequences. This map is then converted to a two-dimensional probabilistic occupancy grid around the car, where each grid element is assigned a probability between 0 and 1 of being occupied. While the occupancy grid provides an estimated distance to detected objects, it tends to have a broader lateral range compared to the point cloud generated by Lidar reference sensors, as illustrated in Figure 1.1. As previously mentioned, ultrasonic sensors offer advantages such as lower production costs and smaller form factors in comparison to reference sensors. Consequently, there is a growing interest within VCC to investigate the potential of enhancing ultrasonic sensor perception through machine learning techniques and leveraging Lidar reference data as a reliable benchmark.

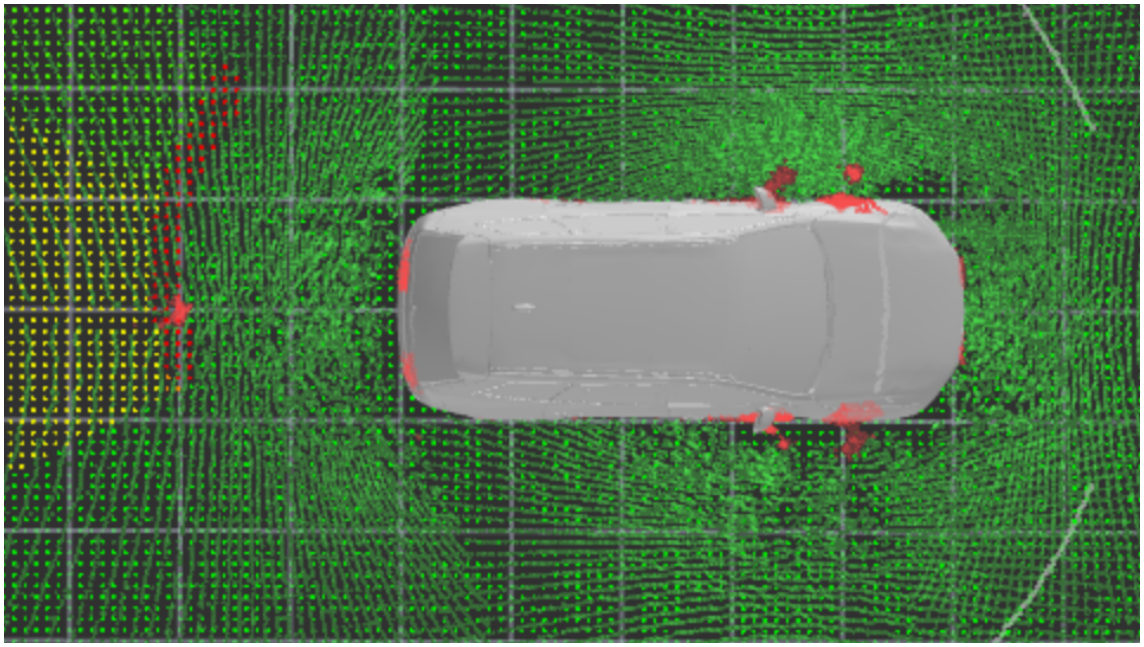


Figure 1.1: Test vehicle with surrounding USS occupancy grid and projected Lidar points. The larger, evenly spaced green, yellow and red points correspond to *free*, *unknown* and *occupied* space respectively, generated by the USS perception system. The smaller, more spread out green and red points correspond to *free* and *occupied* space generated from a projection of Lidar point clouds.

1.1 Objective

In this Master's thesis project we explore the possibility of enhancing the ultrasonic sensor perception of a vehicle. The aim is to train neural networks which use ultrasonic sensor data as input and generates occupancy grids that mimic ground truth grids generated from corresponding short-range Lidar reference data.

When facing any engineering problem one should look for the simplest possible satisfactory solution. With that in mind, an algorithmic approach has already been implemented by VCC, and classical statistical and non-parametric machine learning approaches are deemed to lack the necessary complexity for this task. We therefore

limit ourselves to explore supervised and semi-supervised deep learning solutions.

Using sensor data provided by VCC, we create pairs of USS and Lidar occupancy grids to be used as input and ground truth for our models, respectively. This means that the data in essence is comprised of one-channel images. Drawing from the domains of signal denoising, image segmentation or image-to-image translation, we evaluate the performance of a denoising autoencoder, a deep convolutional image segmentation network called U-Net, and an Image-to-Image conditional Generative Adversarial Network (I2I cGAN).

1.2 Contributions

Together with VCC we develop a comprehensive evaluation framework designed to assess the accuracy and reliability of occupancy grids generated from ultrasonic sensor data in the context of car safety. The framework utilizes a range of classification-based metrics and a distance metric in order to quantify the degree of agreement between generated occupancy grids and a baseline consisting of input USS occupancy grids. This framework facilitates an analysis of the effectiveness of the relevant deep learning models in enhancing the perception of the surrounding environment.

We find that the performance of all networks is highly dependent of the data representation, with only the U-Net outperforming the baseline occupancy grid on one of the data sets. When using a basic representation of a detection consisting of only one point at the detection distance from a sensor, none of the models are able to produce a realistic and accurate perception of the surrounding environment. If we add additional information in the shape of an arc consisting of the possible locations of the detection instead of a point however, the U-Net convincingly outperforms the baseline and all other models in producing a more plausible perception.

Each state of occupied, free-space (and unknown in the case of dense data) in our occupancy grid representations has a distinct value. There is an imbalance between these states, as occupied cells generally only cover a small region of each occupancy grid. To remedy this, we weight each occupied cell of each ground truth grid according to the imbalance between the states in the data set. In the case of our generative model, the output from the discriminator is a downsampling of the input consisting of a Sigmoid classification. We propose a novel pooling network without trainable layers, which pools the weights of a ground truth grid so that they can be used for weighting the output classifications of the discriminator.

Finally, we have developed robust preprocessing pipelines to generate new, accurate representations of data that are extendable and scalable. These will aid in further developments of deep learning solutions for enhancing USS perception at VCC.

1.3 Related Work

The related work here described is based on representing the ultrasonic sensor and Lidar data as two-dimensional occupancy grids, which in turn can be seen as one-channel images. As such, this work mainly concerns techniques using image data.

Denoising autoencoder models, first proposed by Vincent et al. [3], have emerged as effective tools for reducing noise and enhancing the quality of data, particularly in the domain of image processing and restoration. These models are a variant of traditional unsupervised autoencoder models, with the primary objective to reconstruct, clean and denoise versions of corrupted input data by using clean data as ground truth.

The data utilized for training in existing literature predominantly originates from the same domain. To the best of the authors' knowledge, there is no prior work that has utilized denoising autoencoders specifically for transforming input data between different domains. However, it is conceivable to leverage denoising autoencoder models by considering ultrasonic sensor data as a variant of noisy Lidar data. If valid, this approach would possibly allow the generation of data that exhibits closer alignment with the ground truth.

Image segmentation is a fundamental task in computer vision and image processing, which aims to assign a label or category to each pixel or region within an image, effectively distinguishing different objects or areas of interest. In recent years, deep learning techniques have revolutionized its approach and performance by utilizing convolutional neural networks (CNNs) where the network learns to classify each pixel or region based on the visual patterns it encounters. One notable such network is the U-Net proposed by Ronneberger et al. [9], used for biomedical image segmentation. The U-Net employs a contracting and expansive structure, like an autoencoder model, with back connections. These skip connections help to converge the training process and is further discussed in Section 2.1.1.1. If one instead views the pixels of a Lidar image as the correct labels for an ultrasonic sensor image, one might leverage a U-Net to instead generate an image where pixels not previously part of a region are reclassified based on patterns between the input data and ground truth.

Image-to-image translation is a research area within computer vision focused on the task of mapping an input image from a source domain to a target domain while maintaining the semantic content and characteristics of the source domain. A notable aspect of image-to-image translation is its ability to learn mappings between different visual domains without requiring explicit pairwise training examples. Isola et al. [22] use this property to develop the I2I cGAN, where a generator agent is fed with a prior noise distribution conditioned on a set of source images. The generator implicitly learns a mapping to a set of target images by use of a discriminator agent, which simultaneously learns to differentiate between generated and target images and subsequently punishes the generator when it classifies an image correctly. A number of new models based on and related to their work have since been proposed,

including the CycleGAN, the Disco-GAN and the StarGAN [18, 15, 20].

Based on this previous work, it is possible that an I2I cGAN could be used to translate ultrasonic sensor images to Lidar images where these have been explicitly paired. However, a critical question lies in which semantic content and characteristics that the generator preserves. This question can best be answered by applying the theory and observing the results.

1.4 Thesis Outline

To provide a clear roadmap, this thesis is organized into a number of key chapters, following the process of the work.

In Chapter 2 we build up the theory behind the methods most suitable to achieve the thesis objective. We begin by describing artificial neural networks and move on to give a description of convolutional neural networks and autoencoder models. This is followed by a slightly more in-depth theory behind regular and conditional generative adversarial networks. A motivation of how conditional generative adversarial networks fit in the context of image-to-image translation is then given in Section 2.2.3. Finally, a theory of ultrasonic perception is provided to give an understanding of the data used to train the models.

In Chapter 3, we describe the data collection in terms of available size and the logging methods for both USS and Lidar sensors. Furthermore, the choice of data representations are described as well as the procedure for creating these artifacts. We conclude the chapter by presenting the final data sets as well as giving an analysis on the label imbalance.

Chapter 4 is dedicated to explain the evaluation framework and also the three different problem approaches. The main purpose is to motivate the design of both the assessment scores as well as the model architectures and objective functions.

Model evaluation results and sample predictions are presented in Chapter 5 together with a discussion of the taken approach. Finally, drawn conclusions and thoughts on future work are brought up in Chapter 6.

2

Preliminaries

The chapter presents theory and concepts used in this thesis. The first section consists of artificial neural network structures, such as feed forward networks, convolutional networks and autoencoders. The second section provides a short description of generative models followed by an exposition of generative adversarial networks. Finally, the third section gives an overview of the physics of ultrasonic sensors in order to provide an understanding of the data utilized in this thesis.

2.1 Artificial Neural Networks

An artificial neural network or just Neural Network (NN) is a trainable system of computation units inspired by the structure of a mammalian brain. The fundamental building block in an idealised brain is a nerve cell, also called neuron, which has a number of incoming and outgoing connections to other cells. By changing and tuning these connections between the vast amount of neurons, the brain can learn new patterns. A neural network mimics this learning enabling behaviour by updating connection weights between its artificial neurons, called McCulloch-Pitts neurons, as new information is introduced to the NN. These networks are able to learn patterns which makes them effective at information processing tasks such as visual object recognition, machine translation and realistic image generation [25].

2.1.1 Feed Forward Networks

One of the most common network architectures is the Feed Forward Neural Network (FFNN), which structures the McCulloch-Pitts neurons in a layered fashion. In addition, the link structure between the layers do not allow any cyclic connections, thus forcing the signals to flow in one direction. Furthermore, each neuron in a layer are connected to every neuron in the output layer, which is why these layers are referred to as fully connected or dense. The FFNN structure can be expressed as a function

$$\boldsymbol{x} \mapsto f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{\Theta}),$$

where $\boldsymbol{x} \in \mathbb{R}^N$ is the network input which maps to an output $\hat{\boldsymbol{y}} = f(\boldsymbol{x}) \in \mathbb{R}^M$ that depends on the network weight matrix $\boldsymbol{W}$ and bias vector $\boldsymbol{\Theta}$. The function f is a

composite function of each layer, namely

$$f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{\Theta}) = (\ell^{(L)} \circ \ldots \circ \ell^{(1)})(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{\Theta}) \tag{2.1}$$

for an L layer deep network. The kth function can be defined as

$$\ell^{(k)}(\ell^{(k-1)}) = g^{(k)}(\boldsymbol{W}^{(k)}\ell^{(k-1)} - \boldsymbol{\Theta}^{(k)}), \quad \forall k = 1, \ldots, L,$$

where $g^{(k)}$ denotes an activation function, $\boldsymbol{W}^{(k)}$ is the weight matrix, $\boldsymbol{\Theta}^{(k)}$ is the bias weights and the initial layer is given by

$$\ell^{(0)} = \boldsymbol{x}.$$

Each layer ℓ contains a varying amount of McCulloch-Pitts neurons, which decides the shape of $\boldsymbol{W}$ and $\boldsymbol{\Theta}$. Furthermore, the weights and biases are initially set to random values in a suitable range. A commonly used activation function is the **Rectified Linear Units** (ReLU) function, defined as

$$\mathrm{ReLU}(x) = \max(0, x), \quad x \in \mathbb{R}.$$

An alternative to ReLU is the **logistic Sigmoid** function, which is favourably used at the output layer of binary classification networks, namely

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \quad x \in \mathbb{R}.$$

The goal of a NN is to learn target patterns $\mathcal{D}_y = \{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(n)}\}$ from a corresponding input set $\mathcal{D}_x = \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)}\}$ by tuning the weights $\boldsymbol{W}$ and biases $\boldsymbol{\Theta}$ with regards to a loss function $\mathcal{L}$. Hence, the training of a neural network can be seen as an optimization problem with regards to the weights, namely

$$\min_{\boldsymbol{W}, \boldsymbol{\Theta}} \mathcal{L}(f(\boldsymbol{x}^{(\mu)}; \boldsymbol{W}, \boldsymbol{\Theta}), \boldsymbol{y}^{(\mu)}), \quad \forall \mu = 1, \ldots, n.$$

One iterative method of adjusting $\boldsymbol{W}$ and $\boldsymbol{\Theta}$ is by using the gradient of the loss to update each weight element accordingly to

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \alpha \underbrace{\nabla_{\boldsymbol{W}_t} \mathcal{L}(f(\boldsymbol{x}^{(\mu)}; \boldsymbol{W}_t, \boldsymbol{\Theta}_t), \boldsymbol{y}^{(\mu)})}_{\delta \boldsymbol{W}_t^{(\mu)}}, \quad \forall \mu = 1, \ldots, n, \tag{2.2}$$

and

$$\boldsymbol{\Theta}_{t+1} = \boldsymbol{\Theta}_t - \alpha \underbrace{\nabla_{\boldsymbol{\Theta}_t} \mathcal{L}(f(\boldsymbol{x}^{(\mu)}; \boldsymbol{W}_t, \boldsymbol{\Theta}_t), \boldsymbol{y}^{(\mu)})}_{\delta \boldsymbol{\Theta}_t^{(\mu)}}, \quad \forall \mu = 1, \ldots, n, \tag{2.3}$$

where α is the learning rate and t is the current time step. The method is known as **Gradient Decent** (GD) and there exists many variations of it. Optimizing the network weights in a feed forward structured network with a GD like method is called **backpropagation**. It is possible to backpropagate on a subsequence of patterns in a single update to improve stability and training time. We do this by **mini-batching** the patterns in Equation (2.2) according to

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \frac{\alpha}{m_b} \sum_{\mu=i+1}^{i+m_b} \delta \boldsymbol{W}_t^{(\mu)}, \quad \forall i = 0, m_b, 2m_b, \ldots, n - m_b,$$

where m_b is the batch size [25]. Similar mini-batching updates can be performed in Equation (2.3) as well.

2.1.1.1 Boosting Training Performance

There are some additional steps to consider when training a NN with backpropagation to boost network performance and shorten training time. A common issue is to overfit the model to training data, which means that the neural network cannot predict unseen inputs correctly. To acquire a more generalized model, we take a part of the training set and use it to validate the performance of the network after each training epoch. When the loss of the validation set increases, training is stopped early. This framework is known as **early stopping** [25].

Another training issue arises when a parameter in a previous layer changes and affect the remaining weights downstream, which is referred to as internal covariate shifts. This phenomenon causes instability in training and forces lower values of learning rates and increased hyperparameter sensitivity. Mentioned backpropagation instabilities can be decreased by normalizing the layer input batch, which is known as **batch normalization**. We do this by transforming and scaling the input to have zero mean and unit variance, thus reducing the effects of internal covariate shifts [7].

Two more tactics to regularize the training process is to introduce **dropout** and **skip layers** in the network. The dropout function prevents overfitting by randomly disregarding a fraction of q neurons in each hidden layer during training. Remaining $(1-q)$ neurons are active as usual. On the other hand, a skip layer is introduced to oppose the vanishing gradient problem that occurs in deep networks. In short, vanishing gradients is a backpropagation problem that is caused by the chain structure of a NN as seen in Equation (2.1). The closer a layer is to the input, the smaller the gradient change becomes and thus slows down the convergence process [25].

2.1.1.2 Adam Optimizer

A robust stochastic gradient decent based optimizer is the Adaptive Moment Estimation method (Adam). The algorithm iteratively updates exponential moving averages of the mean m_t and variance v_t of the loss gradient $\delta \boldsymbol{W}_t$. The initial step involves updating the biased first and second moments in accordance with

$$\begin{aligned} m_{t+1} &= \beta_1 m_t + (1-\beta_1)\delta \boldsymbol{W}_t \\ v_{t+1} &= \beta_2 v_t + (1-\beta_2)(\delta \boldsymbol{W}_t)^2, \end{aligned}$$

where $\beta_1, \beta_2 \in [0, 1)$ controls the decay rate of the moving averages. Typical values of the hyperparameters are $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The bias-corrected moments are then calculated as

$$\begin{aligned} \hat{m}_{t+1} &= m_{t+1}/(1-(\beta_1)^{t+1}) \\ \hat{v}_{t+1} &= v_{t+1}/(1-(\beta_2)^{t+1}). \end{aligned}$$

Next iteration of weights can now be updated with

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \alpha \frac{\hat{m}_{t+1}}{\sqrt{\hat{v}_{t+1}} + \epsilon},$$

where α is the learning rate and ϵ is a small constant used for numerical stability. The proposed default values of the small constant and learning rate are $\epsilon = 10^{-8}$ and $\alpha = 0.001$, respectively [16].

2.1.2 Convolutional Neural Networks

Another neural network type is the Convolutional Neural Network (CNN), which is favourably used on image classification and object detection tasks. These networks have a similar feed forward structure as the FFNN, which enables it to learn through backpropagation. However, there exists fewer connections between layers in a CNN compared to a dense network. This means that the convolution based network is cheaper to train and is therefore more robust against overfitting to the training data. There are two main types of layers in a these image networks: convolution layers and pooling layers [25].

2.1.2.1 Convolution Layer

Convolutions can be performed on inputs of any dimension, however a common two dimensional convolution will be explained for simplicity. The layer creates a mapping by letting a kernel act on the input matrix, thus creating a downsampled representation to be processed by the corresponding McCulloch-Pitts neuron. Each neuron ℓ_{ij} in the hidden layer $\ell \in \mathbb{R}^{N \times M}$ performs a discrete convolution

$$\ell_{ij} = g\left(\sum_{p=1}^{P}\sum_{q=1}^{Q} w_{pq} x_{p+s_1(i-1),q+s_2(j-1)} - \Theta\right),$$

where w_{pq} are the elements in the weight kernel $\mathbf{W} \in \mathbb{R}^{P \times Q}$, Θ is the bias and s_1, s_2 are integers deciding the **stride** of the filter window. We can also **pad** the input matrix with rows and columns of zeros to ensure that all elements are processed. A convolution layer can use several filters to capture more details and thus create multiple feature maps or **channels**. Such an convolution neuron is described by

$$\ell_{ijk} = g\left(\sum_{p=1}^{P}\sum_{q=1}^{Q} w_{pqk} x_{p+s_1(i-1),q+s_2(j-1)} - \Theta_k\right),$$

where $\ell \in \mathbb{R}^{N \times M \times K}$ and $w_{pqk} \in \boldsymbol{W} \in \mathbb{R}^{P \times Q \times K}$ is an element in the kernel tensor [25].

2.1.2.2 Transpose Convolution Layer

In addition to traditional downscaling convolution layers there exists upscaling Transposed Convolution (TConv) layers, also known as fractionally strided convolutions. If a TConv function $\tilde{\ell}$ with a kernel size of $P \times Q$ and stride 1×1 is applied to an input $\boldsymbol{x} \in \mathbb{R}^{N \times M}$, the resulting dimension of the feature maps $\tilde{\ell}(\boldsymbol{x})$ becomes $(N + P - 1) \times (M + Q - 1)$. Such trainable upscaling layers can be used in different CNN architectures like autoencoders, which are further described in Section 2.1.3. Furthermore, transposed convolutions reverses the forward and backward

passes of a regular convolution. This can be achieved by creating a matrix $\boldsymbol{C}$, which contains all flattened convolution operations performed by a weight kernel $\boldsymbol{W}$, and simply transposing it. Thus, $\boldsymbol{C}^T\boldsymbol{x}$, where $\boldsymbol{x}$ is the flattened input, will generate a flattened output with larger dimension than $\boldsymbol{x}$ [21].

2.1.2.3 Pooling Layer

A pooling layer operates like a convolution layer, except it only compresses the information in the input matrix. Hence, there are no trainable weights or thresholds in these layers. Examples of pooling operations are **max pooling**, which outputs the maximum element in the kernel, or **average pooling**, where the average of the filter inputs are calculated [25].

2.1.3 Autoencoders

An AutoEncoder (AE) is a feed forward type of network which is trained unsupervised to recreate its input, meaning that a target set $\mathcal{D}_y$ is not needed. The network is divided into two parts, an encoder and a decoder. The encoder is either a fully connected or convolutional network where each layer has less neurons than its input layer. The final layer is the so called bottleneck layer with dimension $M << N$, where M is the number of neurons in the bottleneck and N is the input dimension. Let $f_e(\boldsymbol{x})$ denote the encoder part of the network and let $\boldsymbol{z}$ be the latent variables produced by the non-linear mapping. These variables are compressed representations of the input and are the source of the architecture's enhanced learning ability. The decoder, denoted $f_d(\boldsymbol{z})$, takes the bottleneck variables as input and learns to invert the mapping back to $\boldsymbol{x}$ [25]. In other words, the aim of the network is to tune its weights with backpropagation to achieve

$$\boldsymbol{x} = f_d(f_e(\boldsymbol{x})).$$

The autoencoder structure can also be applied to supervised learning for denoising purposes. This class of autoencoders are called **Denoising AutoEncoders** (DAE) and uses a target set $\mathcal{D}_y$ containing patterns

$$\boldsymbol{y}^{(\mu)} \sim \mathcal{Q}(\boldsymbol{x}^{(\mu)}), \quad \forall \mu = 1, \ldots, n,$$

where $\mathcal{Q}$ is a noise distribution [3].

2.2 Generative Models

Generative models are a class of statistical models that, given an observable variable $\boldsymbol{X}$ and a target variable $\boldsymbol{Y}$, captures the joint probability $\mathbb{P}(\boldsymbol{X}, \boldsymbol{Y})$. Given the same variables and an observation $\boldsymbol{x}$, a discriminative model captures the conditional probability $\mathbb{P}(\boldsymbol{Y}|\boldsymbol{X} = \boldsymbol{x})$ [35].

2.2.1 Generative Adversarial Networks

Generative Adversarial Networks were first proposed by Goodfellow et. al. [5], building on generative and discriminative modeling. The authors of this paper propose an adversarial framework where a generative model leverages a discriminative model to achieve its goal of generating new data instances, implicitly using the distribution of the original data set. This is done by letting two networks, the generator and the discriminator, compete in a minimax two-player game. The generator tries to generate new data instances as drawn from the distribution of a training data set. The discriminator on the other hand tries to classify this new data as either a real data instance or a generated one. The training procedure for the generator is to maximize the probability of the discriminator classifying the generated data as real, while the discriminator tries to minimize this probability.

Taking image data depicting portraits of people as an example, this corresponds to training a network to be able to draw samples from a "people-distribution", where the samples are realistic portraits of people that never existed.

In the case of generative adversarial networks, the observable variable $\boldsymbol{X}$ represents the data while $\boldsymbol{Y}$ represents the label fake or real, usually represented by binary values: $\boldsymbol{Y} \in \{0, 1\}$. This label is hidden from the generator, which then tries to model the probability $\mathbb{P}(\boldsymbol{X})$, represented by the data distribution $p_d(\boldsymbol{x})$.

Starting with a prior distribution $p_{\boldsymbol{z}}(\boldsymbol{z})$ of input noise variables to the generator network, the network output distribution $p_g(\boldsymbol{x})$ is a result of the mapping $G(\boldsymbol{z}, \boldsymbol{\theta}_g)$, where G is a differentiable function represented by the generator network with parameters $\boldsymbol{\theta}_g$ producing an output $G(\boldsymbol{x})$.

By defining the mapping $D(\boldsymbol{x}, \boldsymbol{\theta}_d)$ for the discriminator network where D is analogous to G except for the output, $D(\boldsymbol{x})$ represents the probability of $\boldsymbol{x}$ originating from the data. In terms of optimization, finding the optimal solution (D, G) corresponds to finding the optimal value of the minimax two-player game with value function $V(G, D)$, namely

$$\min_G \max_D \; V(G, D) = \mathbb{E}_{\boldsymbol{x} \sim p_d}\Big[\log(D(\boldsymbol{x}))\Big] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}}\Big[\log(1 - D(G(\boldsymbol{z})))\Big].$$

A training paradigm proposed by the authors can be found in Algorithm 1 below.

Algorithm 1 Mini-batch stochastic gradient descent for GAN training

for number of epochs **do**
  for k steps **do**
    Sample minibatch of m_b noise samples $z^{(i)}$ from noise prior $p_{\boldsymbol{z}}(\boldsymbol{z})$
    Sample minibatch of m_b data samples $x^{(i)}$ from data distribution $p_d(\boldsymbol{x})$
    Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\boldsymbol{\theta}_d} \frac{1}{m_b} \sum_{i=1}^{m_b} \left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(\left(z^{(i)}\right)\right)\right)\right]$$

  end for
  Sample minibatch of m_b noise samples $z^{(i)}$ from noise prior $p_{\boldsymbol{z}}(\boldsymbol{z})$
  Update the generator by descending its stochastic gradient:

$$\nabla_{\boldsymbol{\theta}_g} \frac{1}{m_b} \sum_{i=1}^{m_b} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)$$

end for

A conceptual overview of the GAN structure is here shown in Figure 2.1. There are two known issues with the generative model, which is further discussed below.

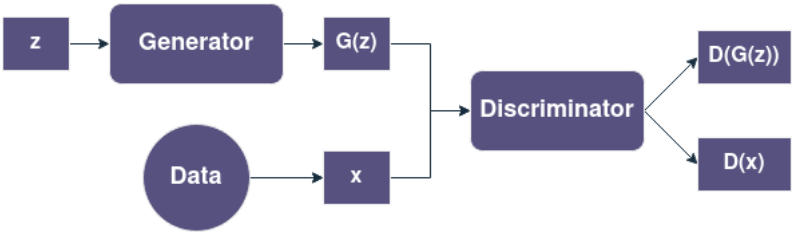


Figure 2.1: Sampled noise is fed to the generator. The discriminator then judges which of a an actual and generated data sample is real and the output is used to calculate the loss for each network.

2.2.1.1 Vanishing Gradient Problems Concerning GANs

Goodfellow et al. [5] note that the generator can fail due to vanishing gradients in the beginning of training, since generated samples are clearly different from the training data at this time. In effect, an optimal discriminator does not provide enough information for the generator to learn. This has later been proven to be the case, and a proposed solution is to train the generator by maximizing $D(G(\boldsymbol{z}))$ instead of minimizing $\log(1 - D(G(\boldsymbol{z})))$ [12].

2.2.1.2 Mode Collapse

The performance of GANs is greatly influenced by the variance present in the training data. This dependency is exemplified by the phenomenon where the generated

samples become highly concentrated within a small region of the discriminator's output space. Consequently, the discriminator can easily distinguish these samples from real data, leading to a limited ability to learn meaningful features from the real data. Effectively, the discriminator is stuck in a sharp local maximum and the generator thus learns to produce only a small set of outputs from the data distribution. In the example of the "people-distribution" above, this corresponds to the generator only producing a limited number of faces, and these might not even look realistic. This of course depends of the nature of the local maximum and when in the training process it occurs.

A proposed remedy to this problem is the Wasserstein GAN (WGAN), which provides stability to the learning process by use of the Wasserstein distance between probability distributions [13]. The application of WGANs in the domain of image-to-image cGANs is relatively new however, and some studies have indicated that the performance improvement achieved by combining these two models may not be very substantial [23].

2.2.2 Conditional Generative Adversarial Networks

The goal of regular GANs is to learn the distribution of a particular data set in order to produce believable samples from this set. Denoting this distribution as the target distribution, one can extend the regular GAN model by conditioning on additional input $\boldsymbol{y}$ to it and thereby mapping a distribution of inputs to the target distribution. The objective function then becomes

$$\min_G \max_D \; V(G,D) = \mathbb{E}_{\boldsymbol{x}\sim p_d}\Big[\log(D(\boldsymbol{x}|\boldsymbol{y}))\Big] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}\Big[\log(1-D(G(\boldsymbol{z}|\boldsymbol{y})))\Big],$$

where both the generator and the discriminator are conditioned on the extra information $\boldsymbol{y}$ in order for G to learn a mapping $G\colon (\boldsymbol{z}|\boldsymbol{y}) \longrightarrow \boldsymbol{x}$. Using this information, one can provide for instance class labels as $\boldsymbol{y}$ to an input image $\boldsymbol{x}$ and thereby segment the distribution $p_g(\boldsymbol{x})$ to produce results corresponding to the provided class label through the adversarial process, where each class label has a distinct maximum in the output space [6].

A conceptual overview of the conditional GAN structure is here shown in Figure 2.2. While this is an overview based on the original paper, many different implementations of cGANs exist and the details of how the inputs are combined are based on the structures of the networks and therefore vary.

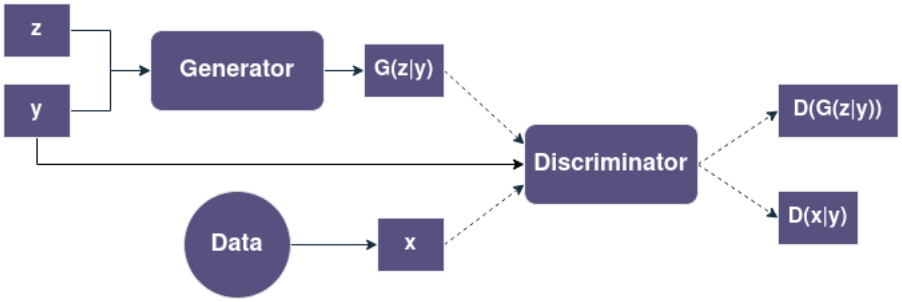


Figure 2.2: Sampled noise is fed to the generator along with some additional information. The discriminator then judges whether an input data sample is real. The output together with the additional information is used to calculate the loss for each network. In contrast with Figure 2.1, only one of the actual and generated data samples is passed to the discriminator (dotted lines).

2.2.3 Conditional GANs for Image-to-Image Translation

In image-to-image translation, given an image $\boldsymbol{x}_A \in A$ from a source domain A, one wants to find a mapping G to an image $\boldsymbol{x}_{AB}$ in a target domain B [28]:

$$\boldsymbol{x}_{AB} \in B : \boldsymbol{x}_{AB} = G_{A \to B}(\boldsymbol{x}_A).$$

This fits well with the definition of a conditional GAN, where G corresponds well to the mapping represented by the generator network, as realized by Isola et al. in [22]. In this paper, the authors investigate conditional adversarial networks as a general-purpose solution to image-to-image translation problems and propose a network structure that has since been adopted and applied to a range of problems. Following their work, in the remainder of this section $\boldsymbol{x}$ represents the conditional information added to the network and corresponds to the source image of domain A. In turn, $\boldsymbol{y}$ represents an instance of the target distribution and corresponds to the target image of domain B. The goal of the generator is thus to learn a mapping $G\colon (\boldsymbol{z}|\boldsymbol{x}) \longrightarrow \boldsymbol{y}$, with the resulting loss function for the GAN:

$$\mathcal{L}_{cGAN}(G, D) = \mathbb{E}_{\boldsymbol{y} \sim p_d}\Big[\log(D(\boldsymbol{y}|\boldsymbol{x}))\Big] + \mathbb{E}_{\boldsymbol{z} \sim p_z}\Big[\log(1 - D(G(\boldsymbol{z}|\boldsymbol{x})))\Big]. \tag{2.4}$$

Further, the authors empirically prove the need for an L_1-regularization by evaluating the results of different compositions of loss functions and networks. This term balances the loss function in Equation (2.4), and is defined as:

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{\boldsymbol{y},\boldsymbol{z}}\Big[\|\boldsymbol{y} - G(\boldsymbol{z}|\boldsymbol{x})\|_1\Big].$$

Here L_1 is used instead of L_2 as the euclidean norm incurs more blurring due to the square on each term. The optimal generator is then given by

$$G^* = \underset{G}{\operatorname{argmin}} \max_{D} \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G).$$

The first term makes sure to produce probablistically realistic output. The second term penalizes the distance between a ground truth image $\boldsymbol{y}$, matched with a conditional input $\boldsymbol{x}$, and a generated output image $G(\boldsymbol{z}|\boldsymbol{x})$. This forces the generator

network to take the conditional input into account also for this term. The parameter λ controls the influence of the L_1-regularization, and thus the balance between the effects of the two terms.

Finally, in order to avoid the vanishing gradient problem related to GANs described above, the loss function for the GAN is further simplified by maximizing $D(G(\boldsymbol{z}|\boldsymbol{x}))$ instead of minimizing $\log(1 - D(G(\boldsymbol{z}|\boldsymbol{x})))$. The resulting objective function for the generator is then

$$\max_G \mathcal{L}_G(G, D) = \max_G \Big\{ \log(D(G(\boldsymbol{z}|\boldsymbol{x}))) - \mathbb{E}_{\boldsymbol{y},\boldsymbol{z}}\Big[\|\boldsymbol{y} - G(\boldsymbol{z}|\boldsymbol{x})\|_1\Big]\Big\}, \tag{2.5}$$

and for the discriminator

$$\max_D \mathcal{L}_D(G, D) = \max_D \Big\{ \log(D(\boldsymbol{y}|\boldsymbol{x})) + \log(1 - D(G(\boldsymbol{z}|\boldsymbol{x})))\Big\}. \tag{2.6}$$

In Equation (2.5) and Equation (2.6), the discriminator and the generator are assumed to be fixed, respectively. This is because training of the network is performed by optimizing both networks interchangeably.

2.3 Ultrasonic Sensors

In acoustics, the wave equation for spherically symmetric pressure fields, or waves, is given by

$$\frac{\partial^2 p}{\partial r^2} + \frac{2}{r}\frac{\partial p}{\partial r} = \frac{1}{c^2}\frac{\partial^2 p}{\partial t^2}, \tag{2.7}$$

where p is the *acoustic pressure*, r is the radius from the origin, c is the speed of sound in air and t is the time of travel for the wave. If the waves are harmonic, the solution to Equation (2.7) can be represented in complex form

$$\boldsymbol{p} = \frac{A}{r}\exp\{j\omega t - kr\},$$

where A is the spherical surface area, ω is the angular frequency and k is the wave number. The actual pressure exerted on the environment is the real part

$$p = \frac{A}{r}\cos(\omega t - kr),$$

with amplitude $P = A/r$. The intensity of this wave is defined according to

$$I = \frac{P^2}{2\rho_0 c},$$

where ρ_0 is and ρ_0 is the equilibrium density. As a result, the intensity of a spherical wave decreases as $1/r^2$ from the origin [2].

Piezoelectricity is the electric charge that accumulates in certain solid materials, such as crystals and certain ceramics, in response to applied mechanical stress.

Modern ultrasonic sensors use a piezoelectic element to create ultrasonic sound waves that expand outwards from the sensor in the shape of a spherical sector, where the shape of the wave front is dependent on solid angle Ω, measured in steradians (Sr). If the wave is reflected, the piezoelectic element vibrates with the same frequency as the reflected wave, generating a signal that can then be leveraged for USS perception, as shown in Figure 2.3.

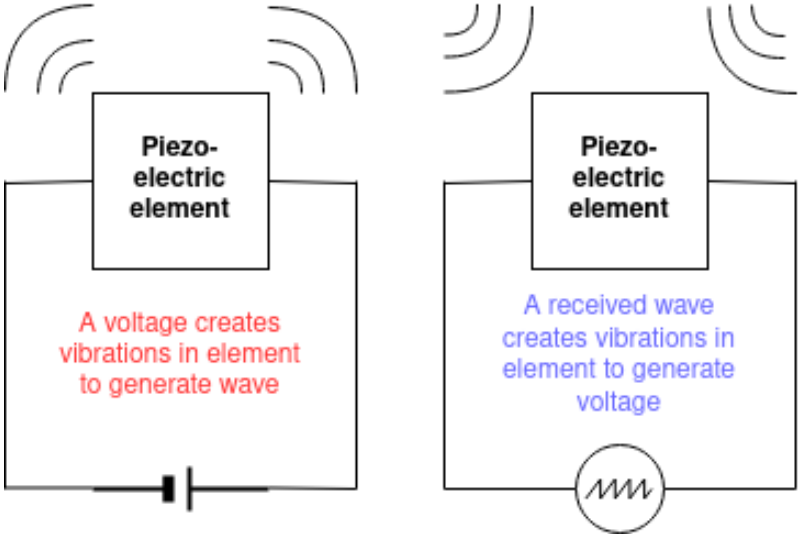


Figure 2.3: A piezoelectic element used for ultrasonic perception.

By analyzing the time it takes for the ultrasonic wave to travel from the sensor to an object and back, the distance can also be calculated based on the speed of the ultrasound. As the intensity of the wave decreases with the distance r, this limits the detection range of ultrasonic sensors to a **max distance**. The sensor membrane also has a refractory period after a wave has been sent or received. As a result, the sensor cannot receive signals, and the corresponding distance an intermediate wave could have travelled during this period without the sensor being able to register it is called the **blind distance** [4].

3

Data

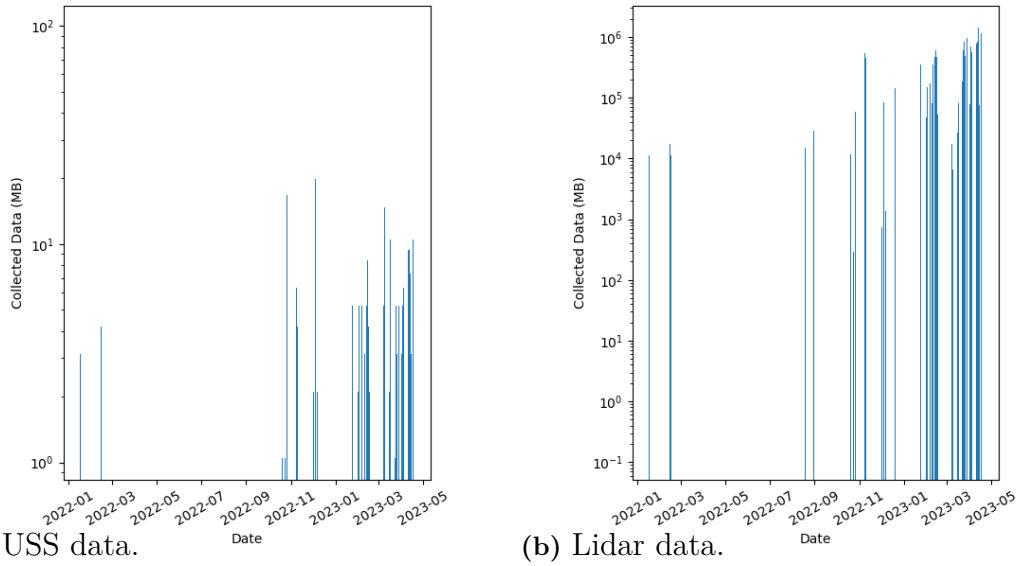
What kind of initial data one is dealing with inevitably influences the choices in data representation available. These choices are in turn linked to model choices where the one influences the other. This chapter describes how we choose our data representations followed by a description of how these representations are created. Chapter 4 in turn, deals with model choices and design.

The first step is to explore the collected data streams to get an intuition of the available quantity and see what is being measured, see Section 3.1. The choice of how to represent the initial USS data is discussed in Section 3.2. Section 3.3 describes what preprocessing and data cleaning steps are used to get the desired representations. The data processing result in three complete data sets. The class balance of these data sets are further investigated in Section 3.4.

3.1 Exploring the Positional Data

The positional USS and Lidar data streams, which will be used for creating training and ground truth data, respectively, are collected using a VCC test vehicle in various locations and circumstances. Some examples are reverse auto-break verification tests, city driving scenarios and garage environments. The data is mainly collected from the Gothenburg region, but also from other locations such as Norway and Spain. Overlapping recordings with both sensor types started in early 2022 and is continuously being collected. At the moment of writing, there are 174 hours or 5000 kilometers of positional logs stored. A majority of the data is captured in low velocities, as shown in Figure 3.2.

The USS data is relatively lightweight and only takes up 459 MB compared to the Lidar streams which are allocated on 22.9 TB. Figure 3.1 shows the various collection dates and sizes of the data streams that VCC has gathered since 2022. Both types of sensor logs are stored in a memory efficient and hierarchical HDF5 format [33], where each measurement stream is chunked into files containing a recording of at most one minute.



(a) USS data. **(b)** Lidar data.

Figure 3.1: Data stream collection amounts per date for **(a)** USS and **(b)** Lidar logs.

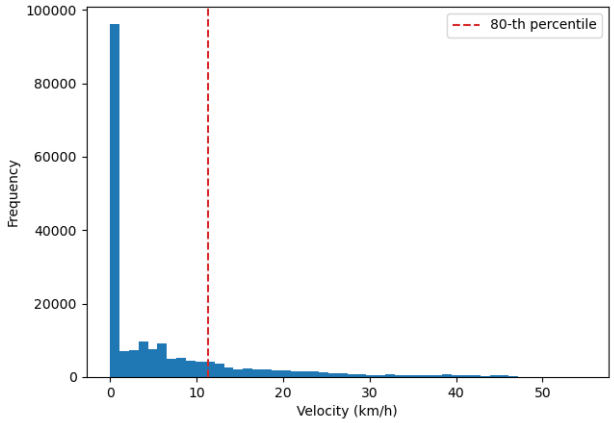


Figure 3.2: Speed distribution of a subsample containing 150000 logs. The 80th percentile velocity is marked with a red, dashed line.

3.1.1 USS Logs

Six ultrasonic sensors are mounted on the front and back bumpers of the test vehicle, respectively, as well as two sensors on each side of the vehicle. Data from the side sensors are not included in this project due to the fact that no combination of USS and Lidar reference data had yet been collected for these positions at the beginning of the project. Figure 3.3 below shows the placement of the sensors as well as the coordinate system used as orientation for sensor firings, where the origin is located at the middle of the rear wheel axle according to the ISO-8855 coordinate system

[31]. Each sensor has a direction angle θ_i indicating its orientation with respect to the x-axis, as well as a resolution angle of $\phi = \frac{\pi}{4}$.

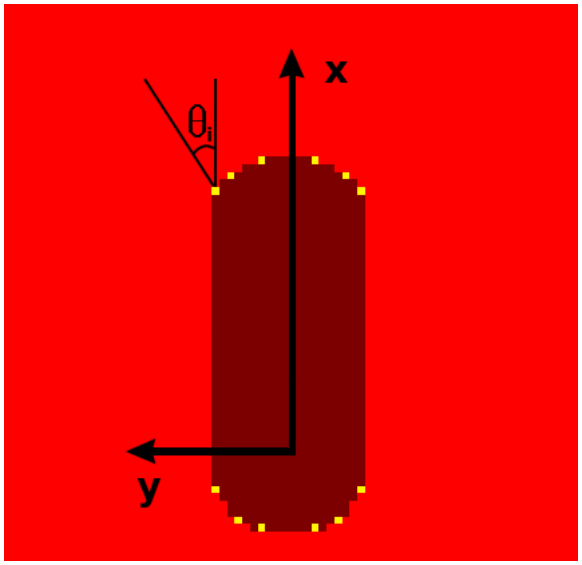


Figure 3.3: VCC test vehicle sensor placements with coordinate system.

Each sensor has a detection range between 100 millimeters to 7 meters and produces a primary and a secondary echo per activation. Sensor readings are done at a frequency of $f = 0.025$ mHz. Each measurement contains information such as the mentioned echo distances, timestamps and other metadata, see Figure 3.4 for the log structure. The USS logs also contain information about the vehicle velocity. The sensor has the capability to process both direct and cross echoes simultaneously, where cross echoes are signal bursts received from neighbouring sensors. Using these cross echoes, one can module five virtual sensors for each bumper, for a total of 11 sensors per bumper. Each signal firing has a sending and a receiving sensor, and information related to a signal firing is therefore divided into a concept called a **signalway**. With the addition of the virtual sensors, there are 16 signalways per bumper. The sensors of each bumper are ordered to fire in sequences, where only a portion of signalways are active, and where all sequences fired constitutes a bumper cycle.

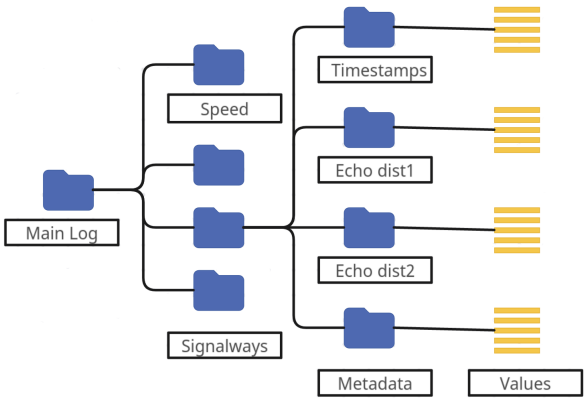


Figure 3.4: File structure of the HDF5 USS logs. The main log contains sensor recordings of one minute intervals. The signalway folders contain the relevant data.

3.1.2 Lidar Logs

The ground truth data is captured with four short range Lidars mounted on the vehicle hood and tailgate as well as in front of left and right side mirrors. This enables the sensors to monitor the environment up to approximately 20 meters away from the test vehicle. A Lidar makes around 40 diode measurements simultaneously at a frequency of 10000 Hz, which are saved in a Packet Capture (PCAP) file. Each entry in the close range sensor logs contain x, y and z coordinates, a UNIX timestamp and other metadata. The four PCAP files are finally stored in a single HDF5 file containing four folders for each of the sensors. The entries are further chunked up into data sets which holds 0.1 second intervals of the measurements, thus creating a number of Lidar *rotations* in each sensor folder. Additionally, a folder with rotation timestamps are available for each Lidar. See Figure 3.5 for a visualization of the file structure.

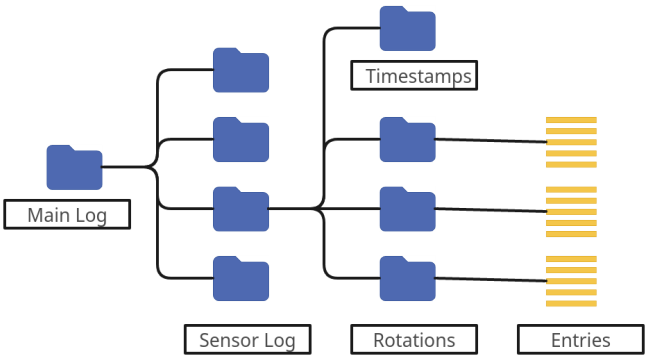


Figure 3.5: File structure of the HDF5 Lidar logs. The main log contains sensor recordings of one minute intervals. The entries contain the positional data.

3.2 Choice of Input Data Representation

The end goal with the project is to train a NN architecture to mimic Lidar generated occupancy grids using USS data, as stated in Section 1.1. However, this objective does not specify the input data representation, which is an essential aspect for the choice of models. It is concluded that two possible input representations can be used by studying the data streams:

1. **Log format** - The first approach strives to use as little preprocessing as possible by using a dense vector as model input. Thus, the USS logs will provide the NN with distance measures and relevant metadata for a given timestamp, which is matched with corresponding ground truth occupancy grid. Considering that there exists a total of 22 virtual and real sensors with approximately 10 fields per sensor, such as amplitude, minimum detection distance, first echo and second echo, the input array will have around 220 elements. The proposed array will be mapped to a 180×128 occupancy grid containing 23040 output pixels, meaning that an upscaling structure would be needed for some networks.

2. **Occupancy grid** - The aim of the second proposal is to produce two dimensional grid inputs, matching the size of the ground truth frames, by fusing USS logs in a time interval and plotting the measured distances. The gathered USS distances can for example be represented as points or arcs centered perpendicular to the sensor position. Since both input and ground truth occupancy grids have matching dimensions, we can consider the mapping task as an image-to-image translation, image segmentation or image denoising problem. There exists off-the-shelf NN models for these machine learning tasks, such as GANs, variational autoencoders and DAEs [28, 32].

The first input scheme is deemed to be less favourable compared with using occupancy grids. There exists upscaling networks for images, e.g. single image super resolution NNs [24], however the problem is rather to map an input vector into a larger output matrix. This upscaling task is not trivial and would require a novel design for some networks, which then might suffer from overfitting problems caused by the small input dimension. The second approach therefore seems more viable as there exists a multitude of established machine learning methods for such frame-to-frame mappings, as stated above.

3.3 Data Creation

With the use of occupancy grids, three input and two suitable ground truth data sets can be created. The input and output frames from respective sets are then matched. A more detailed description of the preprocessing steps taken to generate the data can be found in Appendix C.

VCC:s current representation of the environment is generated through an algorith-

mic approach that produces an occupancy grid consisting of 180×128 cells where each cell corresponds to an area of 0.1×0.1 m^2, occupied cells take on the value of 0, free space cells 1 and unknown cells 0.5. In order to facilitate comparison between results, our USS and ground truth grid representations will share these properties.

3.3.1 USS Occupancy Grids

The most important features provided in the USS data logs is the echo distance, r. Combining this distance with the sensor direction angle θ_i gives the approximate position of a detected object. Due to the wave nature of sound, as described in Section 2.3, this approximate position is located along an arc of length $r\phi$.

Here, a choice in representation appears. With a non-informative prior of where most detections appear along the arc, we could assume that representing a detection by marking only one point at (r, θ_i) as occupied would yield the lowest average error over all detections (as this would be in the middle of the arc). On the other hand the data will be very sparse. In the context of deep learning models, it is generally better for learning if the data contains as little irrelevant features as possible, even if the data in the end will be sparse. This is because the features need to be representative of the underlying data distribution that the model needs to learn [26]. Adding features which are irrelevant could interfere with the learning process by making it more difficult for the model to identify relevant features and patterns.

On the other hand, while this point might be relevant in most cases, marking all points in the arc as occupied is sure to include the true location of the detection. In addition, this would introduce both features that might be irrelevant, but also features that might be present in the ground truth data. In this data representation, we have also allowed for a cell error of 1 in the positive and negative radial direction.

Another point of interest is to explore how the models perform on data containing more, but less relevant information of the environment compared to occupied and free space. We therefore define a third, dense representation containing cells with unknown status corresponding to a value of 0.5.

Our hope with the sparse arc and dense arc representations is that the added information might increase the data diversity, encouraging models to explore the output space and thus to learn a more general and robust mapping to ground truth data representations.

We denote the three different USS data sets by $\mathcal{D}_x^{(\text{point})}$, $\mathcal{D}_x^{(\text{sparse arc})}$ and $\mathcal{D}_x^{(\text{dense arc})}$. For each of the data sets, we screen grids which are either empty or collected at a speed higher than a threshold determined through analysis of positional data logs, see Appendix C.1.4. During the preprocessing we use the information stored in signalways, as well as positions and angles of sensors, to generate the associated detection representation. More details regarding the generation of data sets can be found in Appendix C.

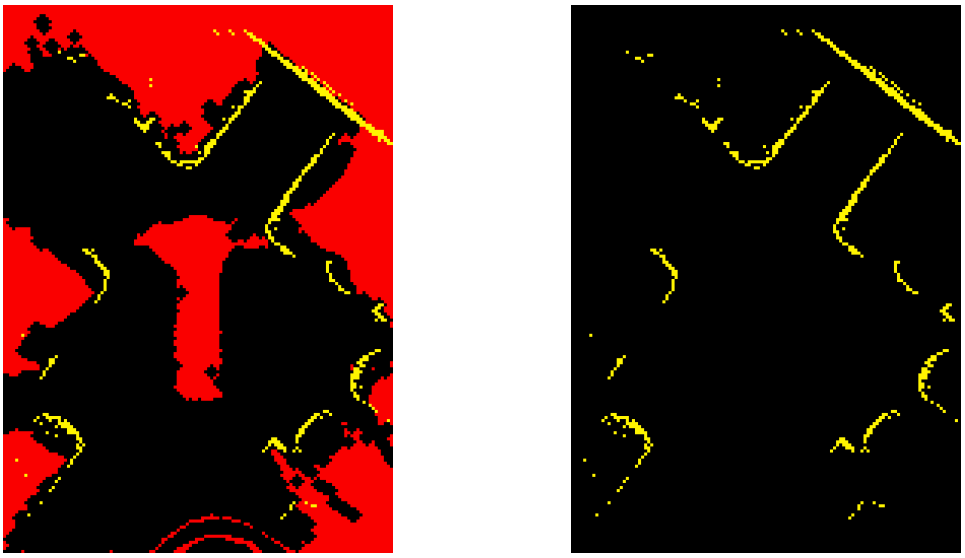
We also mask regions where the sensors are unable to register detections. Cells within the sensor blind distances are marked as free space, while cells outside of the sensor max distances and sensor blind spots are marked as unknown or free space depending on the binary nature of the data set. All three occupancy representations can be found in Figure 3.6.

Figure 3.6: USS data occupancy representations to be used as training data, where yellow denotes occupied cells, black is free space and red is unknown. Left: sparse basic, middle: sparse arc, right: dense arc.

In the rightmost image in Figure 3.6, masking of max distances and blind spots can be seen as red cells in arcs around the vehicle as well as triangular shapes on both sides and areas behind detections, respectively.

3.3.2 Lidar Occupancy Grids

The Lidar Occupancy Grids (OG) can either be modeled as *sparse* or *dense*, see Figure 3.7 for the different representations and also Figure 3.8 for corresponding **fish-eye** images of the parking garage environment. The sparse type contains *occupied* (Yellow) and *free space* (Black) values, thus making the OG a binary bitmap. A new label *unknown* (Red) is introduced in the dense type data to account for areas not having been scanned by any of the short range Lidars. The two Lidar grid representation data sets, sparse and dense, denoted $\mathcal{D}_y^{(\text{sparse})}$ and $\mathcal{D}_y^{(\text{dense})}$, are to be matched with the binary USS representations, $\mathcal{D}_x^{(\text{point})}$ and $\mathcal{D}_x^{(\text{sparse arc})}$, and with the multilabel USS set $\mathcal{D}_x^{(\text{dense arc})}$, respectively.



(a) Dense Lidar OG. **(b)** Sparse Lidar OG.

Figure 3.7: **(a)** Dense and **(b)** sparse Lidar occupancy grids in a garage environment. The yellow, black and red pixels correspond to occupied, free space respectively unknown map labels.

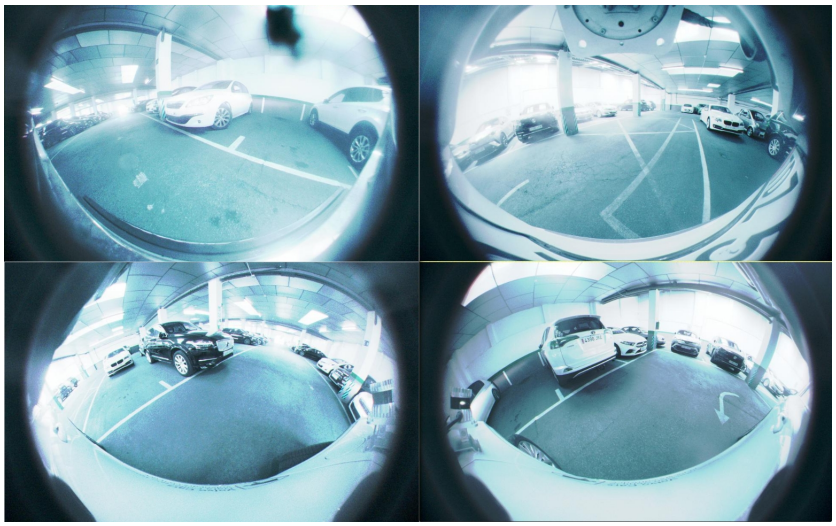
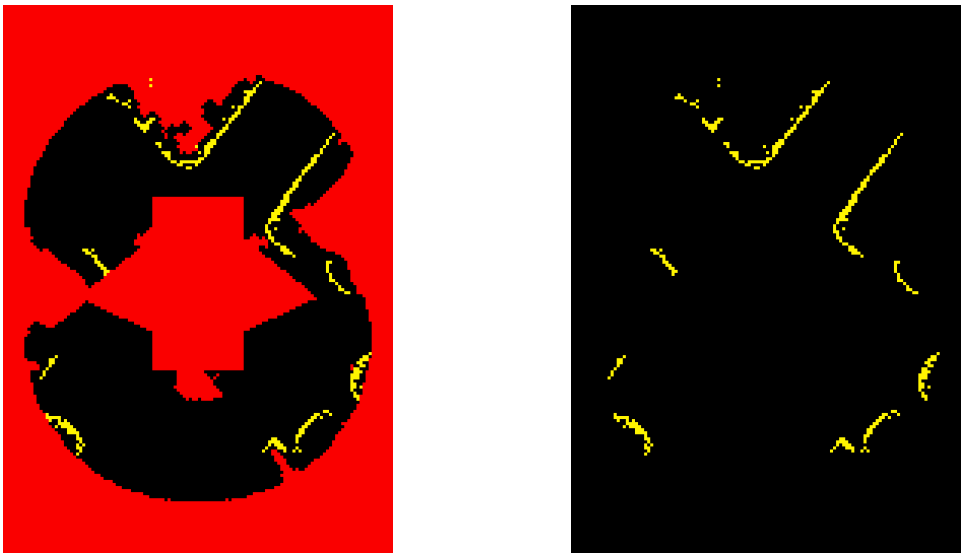


Figure 3.8: Fish-eye camera images of the corresponding environment the Lidar occupancy grids shows in Figure 3.7. Top left window is the front camera, top right is the rear camera, bottom left is the left side camera and bottom right is the right side camera.

The Lidar frames are created by projecting the positional entries from the HDF5 files described in Section 3.1.2 into two dimensions. Filtering is performed on the z-coordinates, which marks each x and y coordinate as ground, obstacle or roof. Corresponding *rotations* are then merged and added to an occupancy grid, as shown in Figure 3.7. Finally, each Lidar grid is masked with the USS max distance as well

as blindspots in order to match the field of vision of the Lidar with that of an ultrasonic sensor, see Figure 3.9. A more detailed description of the Lidar data creation is available in Appendix C.1.2.



(a) Masked dense Lidar OG. **(b)** Masked sparse Lidar OG.

Figure 3.9: Masked **(a)** dense and **(b)** sparse Lidar occupancy grids in a garage environment. The yellow, black and red pixels correspond to occupied, free space respectively unknown map labels.

3.3.3 Data Sets

As mentioned earlier, we have three USS data sets, $\mathcal{D}_x^{(\text{point})}$, $\mathcal{D}_x^{(\text{sparse arc})}$ and $\mathcal{D}_x^{(\text{dense arc})}$, as well as two Lidar sets, $\mathcal{D}_y^{(\text{sparse})}$ and $\mathcal{D}_y^{(\text{dense})}$, which needs to be combined into input and ground truth (GT) pairs. We chose to produce three data sets where USS occupancy grids with binary representations are matched with binary GT and multilabel input with multilabel GT. The created data sets are described in Table 3.1, namely $\mathcal{D}^{(\text{basic})}$, $\mathcal{D}^{(\text{sparse})}$ and $\mathcal{D}^{(\text{dense})}$.

Table 3.1: Description of data set combination to get the input-GT paired data sets.

Data set name	Input set	GT set
Basic	Point	Sparse
Sparse	Sparse arc	Sparse
Dense	Dense arc	Dense

Furthermore, each USS and Lidar frame is paired based on the closest OG timestamps. The merge is valid if and only if the timestamps are close enough in time, otherwise they are disregarded. See Appendix C.1.3 for more details regarding the

merging process and see Appendix C.1.4 for how the threshold was set. A final preprocessing step is to pad the grids from the format 180×128 to 192×128, since it is common practice to repeatedly half the image dimensions in image processing networks.

3.4 Class Imbalance

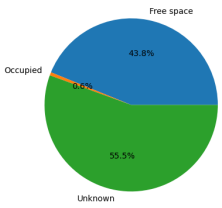
Let us analyse the label distribution of the created data sets, as biases in the data can heavily influence the prediction quality of a model [1]. Figure 3.10 shows the class label distributions of the produced USS and Lidar data sets. We see that the number of occupied pixels is very small compared to the other classes in all sets. In contrast, the free-space and unknown labels in $\mathcal{D}_y^{(\text{dense})}$ and $\mathcal{D}_x^{(\text{dense arc})}$ are quite balanced. To counteract the observed bias in the data, a weighting scheme is proposed, where we let the weight be decided by the inverse proportion of its occurrence [1]. Let n_i be the class occurrence for class i in the data set and C be the set of classes, then the weight for class i is given by

$$w_i = \left(\frac{n_i}{\sum_{k \in C} n_k} \right)^{-1}. \tag{3.1}$$

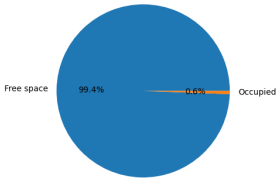
We use (3.1) on $\mathcal{D}_x^{(\text{point})}$, $\mathcal{D}_x^{(\text{sparse arc})}$ and $\mathcal{D}_x^{(\text{dense arc})}$ to produce the class weighting schemes for the labels in respective set as shown in Table 3.2. Note that an exception is made for $\mathcal{D}_x^{(\text{point})}$, as the weights were deemed to be too large. They were therefore multiplied with a factor 0.1.

Table 3.2: Weighting schemes for the three different USS data sets used to counteract the label bias in the data.

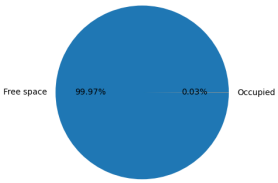
Data set name	Occupied weight	Free space weight	Unknown weight
Point	316.7	1.0	-
Sparse arc	35.9	1.0	-
Dense arc	37.0	2.9	1.6



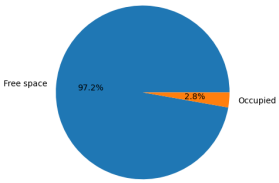
(a) Dense GT set.



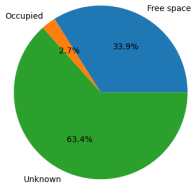
(b) Sparse GT set.



(c) Point USS set.



(d) Sparse Arc USS set.



(e) Dense Arc USS set.

Figure 3.10: Grid label distributions for the **(a)** dense GT, **(b)** sparse GT, **(c)** point USS, **(d)** sparse arc USS and **(e)** dense arc USS data sets.

4

Approach

In this chapter, we begin by developing an evaluation framework in Section 4.1 in order to adequately measure how successfully the models are deployed. We then move on to motivate model choices based on related research domains, give a brief network architecture description and specify design details such as loss functions. In Section 4.2, we present a denoising autoencoder network, followed by a U-Net inspired by related work in image segmentation in Section 4.3. Finally, we present our the image-to-image translation approach Section 4.4, implemented as an I2I cGAN with a U-Net core as generator and a PatchGAN as discriminator.

4.1 Evaluation

An evaluation framework is required to quantify how well the models can reconstruct the ground truth and also measure how generalized the results are. Two different assessment types are proposed, firstly by gridwise errors and secondly by distance deviations. The processed data is split into train, test and validation sets to perform the trials, where the test set is used for model evaluation.

4.1.1 Gridwise Metrics

The gridwise evaluation is inspired by image segmentation assessment frameworks, where each image pixel or grid point is discretely classified [27]. We use two classes; occupied and free-space. This means that the model results must be converted to binary values, since the networks output grid values in the range $[0, 1]$. Therefore, an occupancy threshold $T_{\text{occupied}} = 0.4$ is set, where a predicted grid point $\hat{y}_{ij} \leq T_{\text{occupied}}$ is labeled as occupied and free space otherwise. Each prediction can be labeled as one of the following four classes, since we are considering a binary classification problem;

- **True Positive (TP)** - The predicted pixel is correctly labeled as occupied.
- **False Positive (FP)** - The predicted pixel is incorrectly labeled as occupied.
- **True Negative (TN)** - The predicted pixel is correctly labeled as free-space.
- **False Negative (FN)** - The predicted pixel is incorrectly labeled as free-

space.

These four class labels can be combined to form different informative metrics regarding the quality of the classification. The first metric used is called precision and measures the quality of the positive predictions via the formula

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}.$$

The second metric used is recall, which measures how well the model captures all the true labels. The recall is calculated accordingly to

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}.$$

The third metric used is the F_1-score, which is an aggregation of the precision and recall scores. The formula is the harmonic mean between the two measures, namely

$$F_1 = \frac{2}{\text{Precision}^{-1} + \text{Recall}^{-1}}.$$

The fourth and final score is called accuracy and measures the over all performance of the network by calculating all true predictions by the total number of predictions. Observe that this metric can be misleading in the case of an imbalanced data set. The accuracy score is calculated by the formula

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}.$$

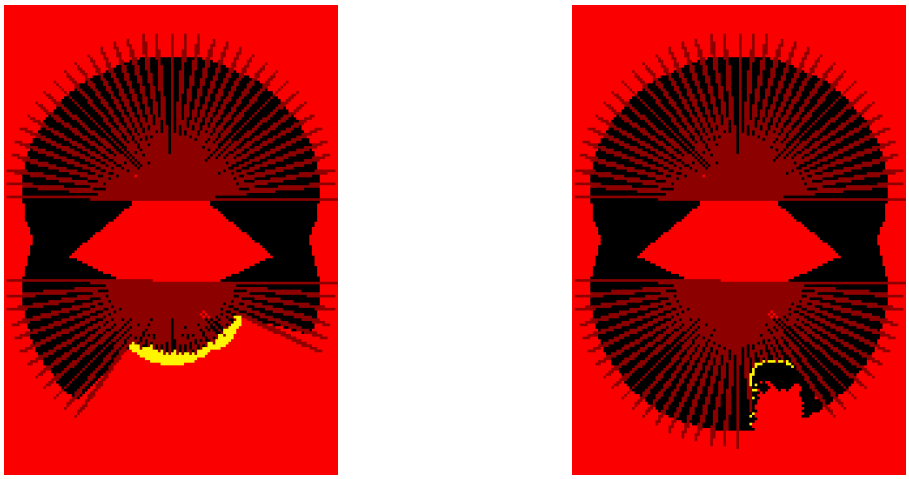
The mentioned classification metrics takes values in the range $[0, 1]$, where 1 is the means perfect performance and 0 correspond to worst possible performance.

4.1.2 Distance Based Metric

An alternative to evaluating errors per pixel is to measure the deviation between a range of predicted and actual distances from the virtual vehicle to the closest occupied cell. We do this by performing a linear search along an angle θ and stepping one pixel forward until an occupied cell is found or if a maximum distance is achieved. Two scans are performed to capture the front and rear environment using the center of corresponding wheel axis, see Figure 4.1. The algorithm increments θ by 5 degrees for each new search iteration until the whole vehicle surrounding has been scanned, see Algorithm 2 for pseudo code. After the entire test prediction and ground truth frames have been evaluated can the Root Mean Squared Error (RMSE) be calculated to get a sense of how much the distances varies. The RMSE can be seen as an estimation of one standard deviation, given that the deviations are normally distributed. The RMSE is defined as

$$\text{RMSE}(\hat{\boldsymbol{d}}, \boldsymbol{d}) = \sqrt{\frac{1}{n_d} \sum_{i=1}^{n_d} (\hat{d}_i - d_i)^2},$$

where n_d is the vector length, $\hat{d}_i$ is the estimation and d_i is the ground truth.



(a) USS frame. **(b)** Lidar frame.

Figure 4.1: Distance measurement algorithm performed on **(a)** a USS frame and on **(b)** a Lidar frame. The burgundy lines represent the linear search that is performed for each 5 degree angle from the rear and front axis.

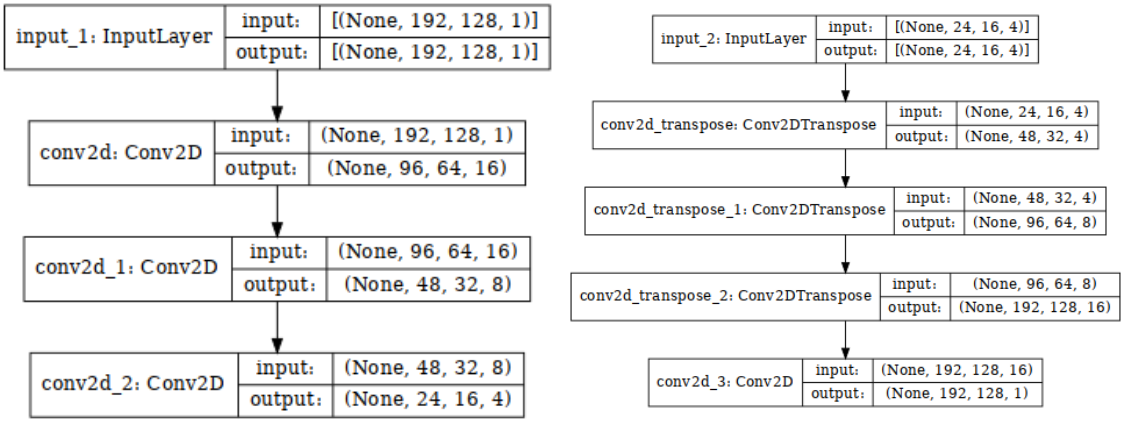
Algorithm 2 Calculate virtual distances

Initialize $\Theta_{\text{rear}} = \{90, \ldots, 270\}$ to contain angels with resolution η
Initialize $\Theta_{\text{front}} = \{-90, \ldots, 90\}$ to contain angels with resolution η
Initialize list $\boldsymbol{d}$
for $\Theta_i = \Theta_{\text{rear}}, \Theta_{\text{front}}$ **do**
    Set start point x_0 in the center of the i vehicle axis
    for $\theta \in \Theta_i$ **do**
        Set $x \leftarrow x_0$
        Set $r \leftarrow r_0$
        while OG value at position x is not occupied and $r < r_{\max}$ **do**
            Increment $r \leftarrow r + 1$
            $x \leftarrow x_0 + [r\cos\theta, r\sin\theta]$
            Truncate element-wise $x \leftarrow \lfloor x \rfloor$
        end while
        Append distance $\|x - x_0\|$ to $\boldsymbol{d}$
    end for
end for
return List $\boldsymbol{d}$

4.2 Denoising Approach

In this approach we see the USS frame as a distorted image of the corresponding Lidar frame, as stated in Section 1.3. The aim is to train a convolutional autoencoder architecture, which is typically used in image denoising tasks [3], in order to perform the mapping between the USS and Lidar grids. The hope is that the encoder will encrypt important properties of the denoising process in the latent variables $\boldsymbol{z}$, as discussed in Section 2.1.3.

The denoising model is inspired by the autoencoder implementation in TensorFlow's article on AEs and DAEs [32]. The encoder architecture consists of three convolution layers and outputs $\boldsymbol{z} \in \mathbb{R}^{24\times16\times4}$. In contrast, the decoder is built by three transposed convolution layers in order to upscale the image. A final convolution layer is added to take the image back to its original dimensions, and thus outputs $\hat{\boldsymbol{y}} \in \mathbb{R}^{192\times128}$. Furthermore, a logistic Sigmoid function is applied to $\hat{\boldsymbol{y}}$ so that the values can be mapped to occupancy probabilities. See Figure 4.2 for an overview of the encoder and decoder architectures and Appendix B.1.1 for detailed network layer information.



(a) Encoder **(b)** Decoder

Figure 4.2: **(a)** Encoder and **(b)** decoder architectures used in the denoising autoencoder model. The encoder contains three downscaling convolution layers, while the decoder contains three upscaling transposed convolution layers with an additional convolution layer with one channel.

Two different loss functions are used for the DAE architecture, one for the dense data set and one for the basic and sparse data sets, respectively. Vincent et al. proposes that autoencoders utilizing real valued inputs can use a **Mean Squared**

Error (MSE) loss function. The MSE is defined as

$$\mathcal{L}_{\text{mse}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{n'} \sum_{i=1}^{n'} (\hat{y}_i - y_i)^2, \tag{4.1}$$

where $\hat{\boldsymbol{y}}, \boldsymbol{y} \in \mathbb{R}^{N \times M}$ and $n' = NM$. The objective function is a reasonable choice since $\mathcal{D}^{(\text{dense})}$ contains values in the range $[0, 1]$. Networks dealing with binary inputs, like the USS grids in $\mathcal{D}^{(\text{basic})}$ and $\mathcal{D}^{(\text{sparse})}$, are better suited with a **Binary Cross Entropy** (BCE) loss [3]. In addition to the BCE loss, a **Mean Absolute Error** (MAE) term with a scalar weighting $\lambda \geq 0$ is added. Thus, the binary objective is given by

$$\mathcal{L}_{\text{bin}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \underbrace{\frac{1}{n'} \sum_{i=1}^{n'} (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i))}_{\text{BCE}} + \lambda \underbrace{\frac{1}{n'} \sum_{i=1}^{n'} |\hat{y}_i - y_i|}_{MAE}. \tag{4.2}$$

This objective is inspired by the one used for image-to-image conditional GANs. Adding a more traditional loss term, such as MAE, may improve results, which is why (4.2) is used [10]. To distinguish between the two DAEs using loss (4.1) and (4.2) the models are denoted as $\text{DAE}^{(\text{mse})}$ and $\text{DAE}^{(\text{bin})}$, respectively. Finally, there exists a class imbalance between the occupied, free space and unknown labels, as described in Section 3.4. We therefore implement the weighting scheme proposed in Table 3.2 for a more unbiased training process of $\text{DAE}^{(\text{bin})}$. This is possible since the model considers discrete binary labels in contrast with $\text{DAE}^{(\text{mse})}$.

4.3 Image Segmentation Approach

As an alternative to the denoising approach, we propose to see the problem as an image segmentation problem. The discrete classes occupied, free space and unknown are used instead of seeing each pixel as a value of 0, 0.5 or 1. The goal is to learn a mapping between the mentioned input and ground truth grid labels. For this purpose is a U-Net architecture suitable. The CNN model was originally proposed for biomedical image segmentation and uses skip layers to counteract the vanishing gradient problem, as discussed in Section 2.1.1.1. Furthermore, the NN structure consists of down- and upsampling blocks, like the DAEs. In general, each downsample block consists of a batch-normalized convolution layer and the upsample blocks are composed of batch normalized transposed convolution layers with dropout. The U-Net has seven and six down- and upsampling blocks respectively, with skip connections as shown in Figure 4.3.

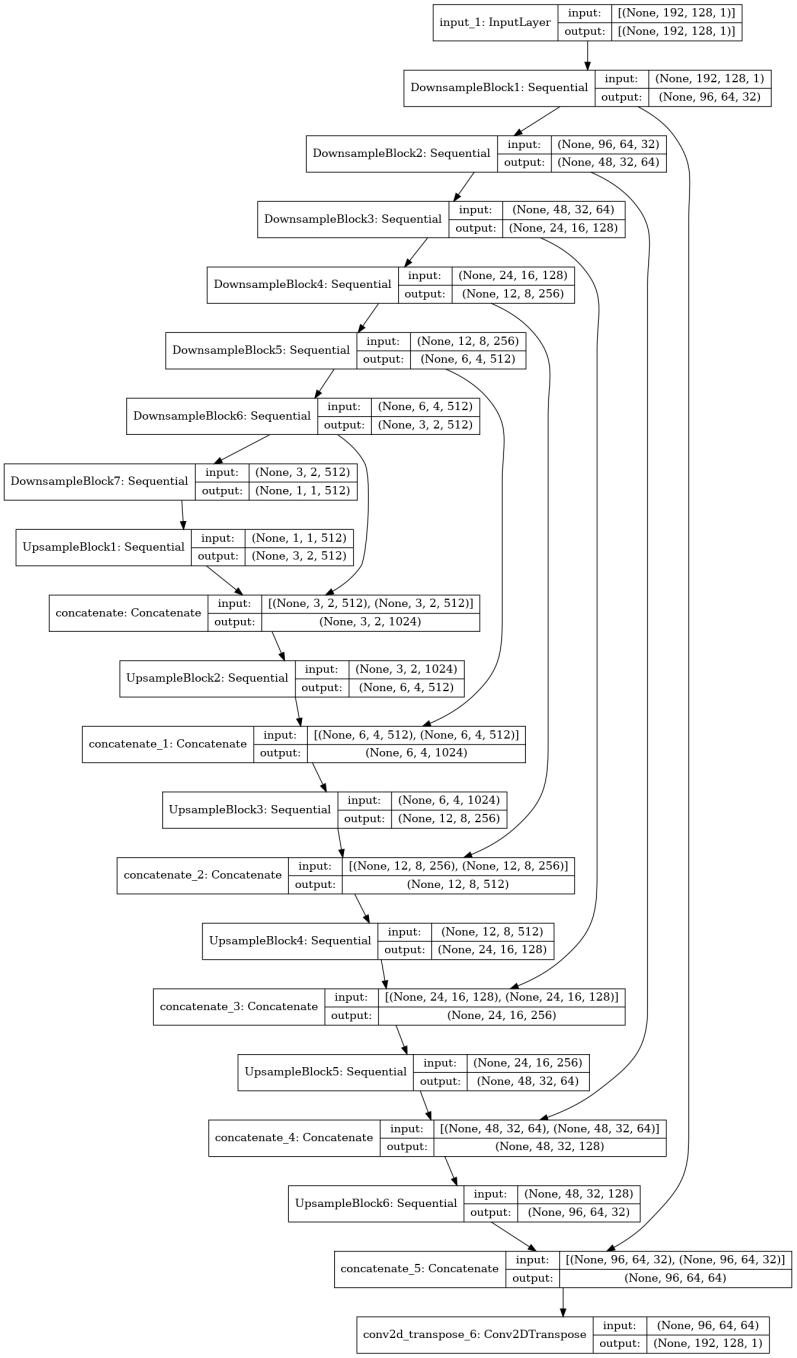


Figure 4.3: U-Net image segmentation architecture, which uses up- and downsampling blocks with skip connections for improved training stability.

The binary loss described in Equation (4.2) is also a suitable choice for the U-Net when trained on $\mathscr{D}^{(\text{basic})}$ and $\mathscr{D}^{(\text{sparse})}$, since we have only two labels. A logistic Sigmoid function is also used on the final model layer for this reason. We denote this model with $\text{UNET}^{(\text{bin})}$. However, another loss function and final layer is required to handle multiple classes, such as in $\mathscr{D}^{(\text{dense})}$. The general **Cross Entropy** (CE)

loss is defined as

$$\mathcal{L}_{\text{ce}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{n'} \sum_{i=1}^{n'} \sum_{k \in C} y_{ik} \log \hat{y}_{ik}, \tag{4.3}$$

where y_{ik} is a binary indicator (0 or 1) if label k is the correct classification for pattern i, and $\hat{y}_{ik}$ is the estimated probability that observation i is of class k. This means that each U-Net output pixel must represent a probability distribution of the three different classes, and thus is the final layer changed to output $\hat{\boldsymbol{y}} \in \mathbb{R}^{192 \times 128 \times 3}$. We denote the multiclass image segmentation model as U-Net$^{(\text{ce})}$. Finally, the class imbalance is taken into account by implementing the weighting schemes proposed in Table 3.2.

4.4 Image-to-image Translation Approach

The approach of using generative models is based on viewing the stated problem as an image-to-image translation problem, as described in Section 2.2.3. The architecture for the generator used in the Image-to-Image translation conditional GAN is the same as in Figure 4.3, with an initial Gaussian noise layer of shape $(192 \times 128 \times 1)$, representing the prior distribution $p_{\mathbf{z}}(\mathbf{z})$. The choice of noise level is based on Wu et al. [17], advocating a use of zero-centered normal noise with a standard deviation of 0.1 for pixels normalized to the range $[0, 1]$. A description of the structure of the U-Net is given in Section 4.3 above, with additional details in Appendix D. The architecture for the discriminator can be found in Figure 4.4.

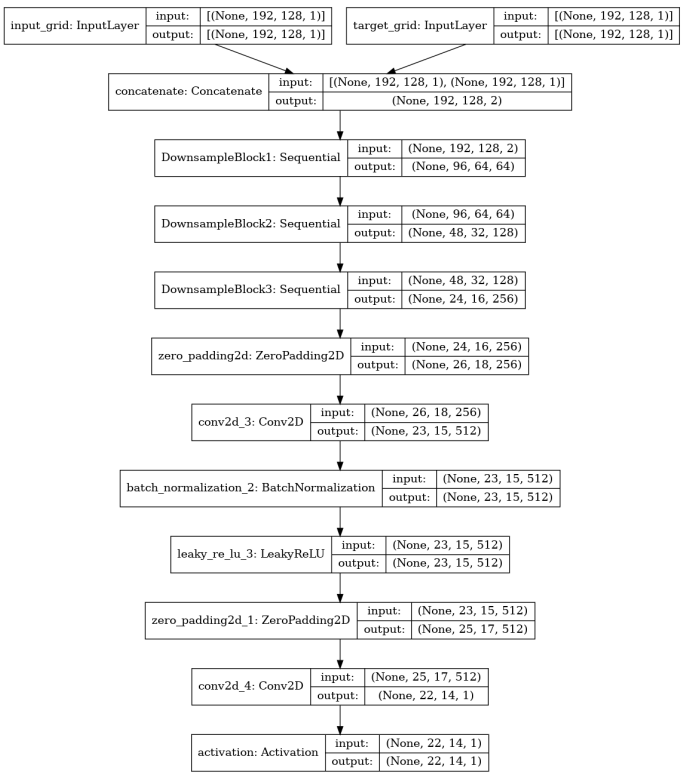


Figure 4.4: Discriminator architecture for the Image-to-Image translation conditional GAN.

In essence, this is a regular convolutional network that takes two inputs. The first input is either the Lidar ground truth grid $\boldsymbol{y}$, or a generated image $G(\boldsymbol{z}|\boldsymbol{x})$, while the second input is the conditional information $\boldsymbol{x}$, i.e. the USS grid.

By processing the input image and the conditional information together, the PatchGAN is able to learn to focus on the parts of the image that are relevant to the given task, such as the texture and color of specific regions of the image. The output differs from a regular ConvNet in that the network does not classify the entire image as real or generated. Instead it generates a grid of patches with size $(22 \times 14 \times 1)$, where each patch corresponds to a range of convoluted pixels from the concatenated input and carries a value corresponding to the probability that it real or generated. This output is then averaged when computing the final loss for the specific input image (or batch of images).

As the sparse USS and Lidar data representations are very similar in many parts of the frames, we introduce a novel way of influencing the probability of particular patches in the output of the discriminator. We weight the data according to the class imbalance described in Section 3.4, as is done in the U-Net image segmentation approach, and then pass a weighted ground truth frame through a sequence of max pooling layers and a final average pooling layer, as illustrated in Figure 4.5.

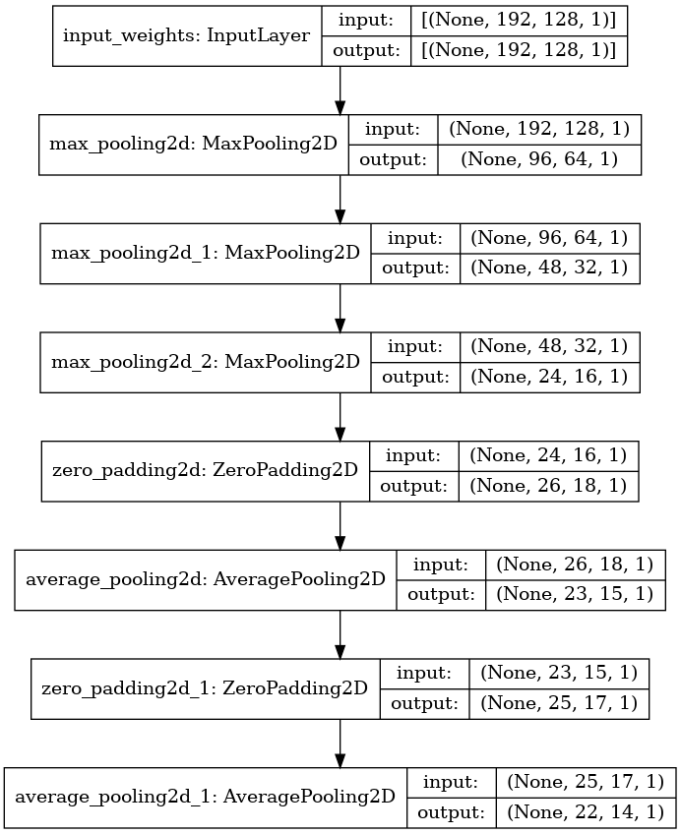


Figure 4.5: Pooling network for sample weights generated from ground truth data.

The result is then used to weight the discriminator output patches, where patches

containing occupied pixels are weighted more heavily. An additional motivation for weighting the discriminator output is that preliminary runs showed that the discriminator entered mode collapse at the beginning of training when on the sparse data representations and as result the generator only learned to produce a blank output with all cells marked as free space. The details of mode collapse are covered in Section 2.2.1. A sample weighting is shown in Figure 4.6.

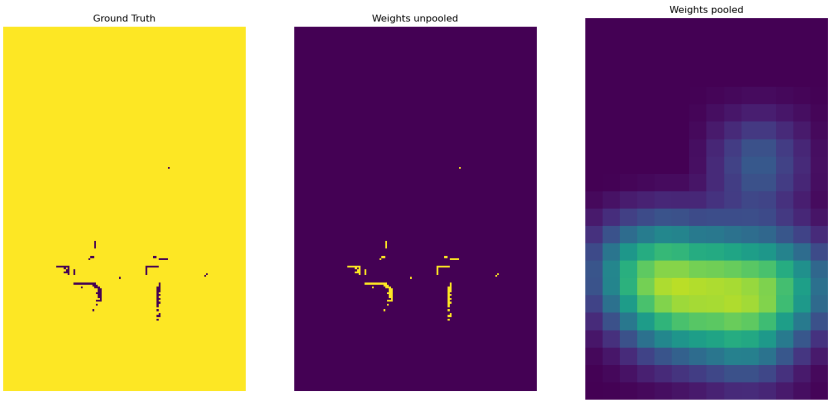


Figure 4.6: Visualization of weighting of patches based on occupied pixels in a ground truth frame, where values range from the maximum weight 0.997 (yellow) and the minimum weight 0.03 (blue).

While the region affected by this weighting is quite rough, as can be seen in the rightmost image, it is an improvement to the previous uniform weighting of patches.

The minimax loss function is in practice generally realized by using the binary cross entropy loss function. This is because training a discriminator using Equation (2.6) is analogous to minimization of a standard cross entropy loss when training a binary classifier with a Sigmoid output, i.e. binary cross entropy, where true labels (1) are used for input from the data distribution, i.e. real images, and false labels (0) are used for generated input. This also applies for the first term in Equation (2.5) for the generator, but where with flipped labels for real and generated images [14].

The generator network and the discriminator are combined into a composite model constituting the entire GAN, as visualized in Figure 4.7, which used to train the generator network.

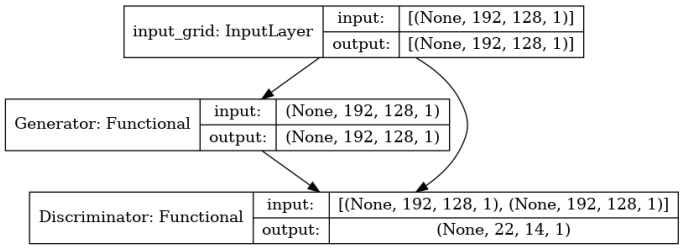


Figure 4.7: Complete Image-to-Image cGAN architecture.

In this model, the discriminator weights are frozen, meaning that they are not updated when performing backpropagation through the network. This is done because the discriminator would otherwise learn from the progress of the generator and any local minimum would then become a moving target, causing the generator to struggle with learning.

The connection between the networks is still important however, as the generator is not directly connected to the cross entropy loss that it is trying to affect – this is instead produced by the discriminator. The gradients obtained when backpropagating through the discriminator network therefore directly affect the weights of the generator network and a connection between the two networks are created as a result. Meanwhile, skip connections in the U-Net assure that vanishing gradients are mitigated.

5

Model Evaluation

The purpose of the thesis is to evaluate how the methods proposed in Chapter 4 perform on the generated data sets $\mathcal{D}^{(\text{basic})}$, $\mathcal{D}^{(\text{sparse})}$ and $\mathcal{D}^{(\text{dense})}$ described in Chapter 3. We start by explaining the experimental setup in Section 5.1 and then follow up with results gathered using the evaluation framework in Section 5.2. Finally, the data representations, approaches and evaluation method are discussed in Section 5.3.

5.1 Training Environment

The preprocessing of the data streams resulted in 534647 USS and Lidar frame pairs for each of the basic, sparse and dense data sets. The frames are split into training, test and validation data according to an 80-10-10 split. This ensures that all models are evaluated on the same data and thus a fair comparison is made. Furthermore, the data in the validation and test sets are taken in a consecutive sequence, meaning that frames coming from a specific time period is ensured to be unseen when evaluated and validated upon. The frame pairs are shuffled internally in each data set to regularize the training. Backpropagation processing is executed on an NVIDIA TITAN RTX GPU, which speeds up training significantly for large machine learning models compared to using traditional CPUs [19]. Python is used to build the training environment using Google's state-of-the-art machine learning framework TensorFlow [8]. Finally, each input data set is also directly evaluated on the ground truth. The resulting metrics can be used as a baseline to compare with the model predictions.

5.1.1 DAE Setup

The DAE setup uses early stopping to avoid overfitting, as discussed in Section 2.1.1.1, where the training is stopped if the validation error is not improved in T_{patience} epochs. Otherwise, the maximum number of epochs is T_{epochs}. Mini-batching is also used with size m_b. Furthermore, the Adam optimizer scheme is used to tune the network parameters with the default parameters except an adaptive learning rate, see Section 2.1.1.2. The initial learning rate α_0 is decreased after n_{step} training iterations with a decay rate β_{decay}. Finally, the weighting schemes proposed in Section 4.2 are implemented for the DAE$^{(\text{bin})}$ model. See Table 5.1 for the hyperpa-

rameter values used to train $\text{DAE}^{(\text{mse})}$ and $\text{DAE}^{(\text{bin})}$.

Table 5.1: Hyperparameters used to train the two denoising autoencoder models $\text{DAE}^{(\text{mse})}$ and $\text{DAE}^{(\text{bin})}$.

Hyperparameter	Symbol	Value
Mini-batch size	m_b	32
Max epochs	T_{epochs}	100
Patience	T_{patience}	8
Initial learning rate	α_0	0.01
Decay steps	n_{step}	15000
Decay Rate	β_{decay}	0.9
MAE weighting (only for bin-loss)	λ	10

5.1.2 U-Net Setup

For simplicity, the same setup and hyperparameters used for the DAE models were also used for the U-Net, see Table 5.1. Furthermore, different MAE loss weightings $\lambda = 0, 10, 50, 100$ were tested for $\text{UNET}^{(\text{bin})}$ when training on $\mathcal{D}^{(\text{sparse})}$ to see how the results were effected. Two additional $\text{UNET}^{(\text{bin})}$ setups were tested, one with smaller mini-batch size $m_b = 16$ and the other with constant learning rate $\alpha = 0.0001$, both using data from $\mathcal{D}^{(\text{sparse})}$. Finally, the weighting schemes discussed in Section 4.3 were implemented for $\text{UNET}^{(\text{bin})}$ and $\text{UNET}^{(\text{ce})}$ respectively, depending on the data set the models were trained on.

5.1.3 I2I-cGAN Setup

The cGAN is trained using an Adam optimizer with a fixed learning rate of $\alpha = 2{\cdot}10^{-4}$ and coefficient $\beta_1 = 0.5$. Just as for $\text{UNET}^{(\text{bin})}$, different MAE loss weightings $\lambda = 0, 10, 50, 100$ were tested. All training is done using a batch size of $m_b = 1$. These parameter values are chosen based on results from preliminary runs as well as the results obtained by Isola et al. [22]. These parameter values are used for every data set.

Goodfellow et al. suggest to use mini batches when alternating the training of the generator and discriminator [5], as can be seen in Algorithm 1. This has no effect when using the above given batch size. As mentioned earlier, the USS and Lidar grids are also very similar, which means that the discriminator might need more data in order to reach a local optimum before switching to training the generator. We therefore choose to train the discriminator and generator using alternating intervals, where each interval corresponds to ~ 5000 grids. This is done a total of 100 times per network, stretching over 2 epochs. The reason for the low number of epochs is because a longer running training showed the losses quite quickly converging to a small range of values. In addition, the cGAN takes a much longer time to train compared to the other models.

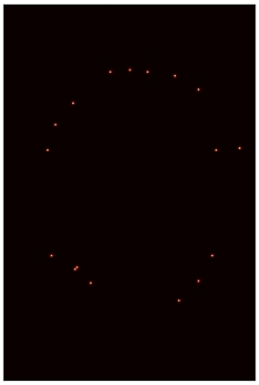
5.2 Results

In this section we present the best performing models from each approach, based on the evaluation framework proposed in Section 4.1. The approaches are deployed on the three data sets mentioned in Chapter 3. All results described in the experiment setup in Section 5.1 are available in Appendix A. Representative sample predictions are shown in addition to the five performance scores to visualize the output. Furthermore, the best measured result per metric is marked in bold in each table.

5.2.1 Basic Data

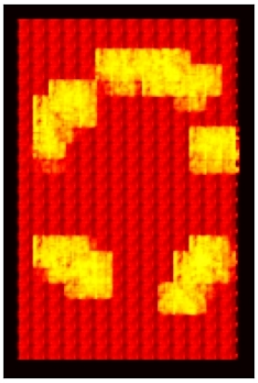
First we consider the models trained on $\mathcal{D}^{(\text{basic})}$, namely $\text{DAE}^{(\text{bin})}_{\lambda=10}$, $\text{UNET}^{(\text{bin})}_{\lambda=10}$ and $\text{cGAN}_{\lambda=10}$. The model evaluation metrics are shown in Table 5.2 and corresponding prediction samples are visualized in Figure 5.1. We start by comparing the denoising and image segmentation results. The $\text{UNET}^{(\text{bin})}_{\lambda=10}$ consistently achieves slightly better performance indicators and creates a more detailed prediction grid opposed to the $\text{DAE}^{(\text{bin})}_{\lambda=10}$ model. The trained autoencoder creates large square artifacts and thus loses prediction precision, see Figure 5.1c and 5.1d. However, both models have a hard time to make qualitative occupancy predictions as they both get a low precision score.

The $\text{cGAN}_{\lambda=10}$ model have issues predicting any occupied pixels, based on the close to zero recall score as well as in the sample frame in Figure 5.1e. Furthermore, a high accuracy metric is achieved since the model has reached a state mirroring the symptoms of mode collapse, discussed in Section 2.2.1.2, where almost only free space are predicted. This might be caused by the vast majority of free space pixels, as shown in Section 3.4. The few positive predictions that are made have better precision than the other models. Finally, no model manages to outperform the baseline on all measurement accounts and the baseline is not improved considering the distance RMSE metric.

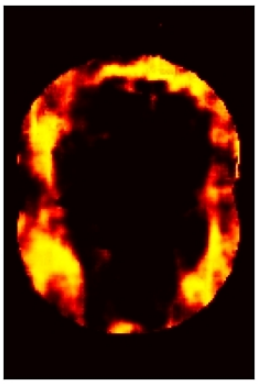


(a) Point USS input.

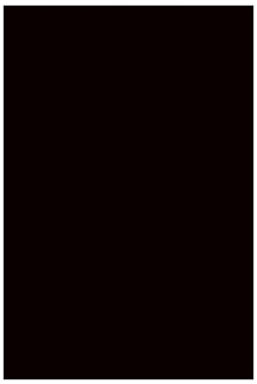
(b) Sparse Lidar GT.



(c) $\text{DAE}^{(\text{bin})}_{\lambda=10}$ prediction.



(d) $\text{UNET}^{(\text{bin})}_{\lambda=10}$ prediction.



(e) $\text{cGAN}_{\lambda=10}$ prediction.

Figure 5.1: Predictions from models trained on $\mathcal{D}^{(\text{basic})}$, namely **(c)** $\text{DAE}^{(\text{bin})}_{\lambda=10}$, **(d)** $\text{UNET}^{(\text{bin})}_{\lambda=10}$ and **(e)** $\text{cGAN}_{\lambda=10}$. We can also see **(a)** the input and corresponding **(b)** ground truth frames. The grids are collected between 09:26:07 and 09:27:07 at 2023-04-05. Furthermore, yellow, red and black denotes occupied, unknown and free-space, respectively.

Table 5.2: Evaluation results from the trained models on $\mathcal{D}^{(\text{basic})}$ as well as a baseline comparison.

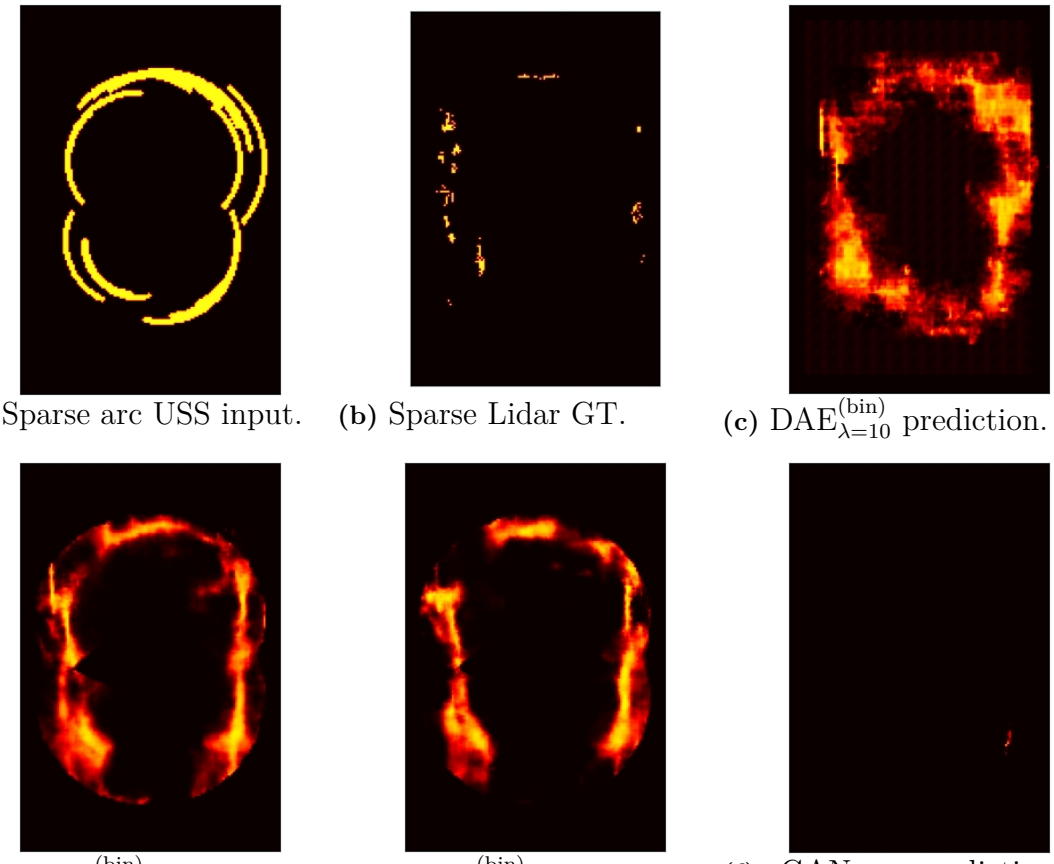
Model	Precision [%]	Recall [%]	F_1-score [%]	Accuracy [%]	Distance RMSE [m]
None	10.04	0.91	1.67	99.11	**1.8428**
$\text{DAE}^{(\text{bin})}_{\lambda=10}$	3.36	68.45	6.41	83.39	2.9694
$\text{UNET}^{(\text{bin})}_{\lambda=10}$	4.90	**71.46**	**9.16**	88.23	2.1974
$\text{cGAN}_{\lambda=10}$	**21.76**	0.02	0.04	**99.17**	1.8569

5.2.2 Sparse Data

The best performing models trained on $\mathcal{D}^{(\text{sparse})}$ are $\text{DAE}^{(\text{bin})}_{\lambda=10}$, $\text{UNET}^{(\text{bin})}_{\lambda=50}$, $\text{UNET}^{(\text{bin})}_{\lambda=100}$ and $\text{cGAN}_{\lambda=50}$. The evaluation metrics and prediction samples from the models

are shown in Table 5.3 and Figure 5.2, respectively. We see that each approach outperforms its counterpart model trained on $\mathcal{D}^{(\text{basic})}$, considering the F_1-score, accuracy and distance RMSE. However, some tendencies observed in Section 5.2.1 are still remaining. For example, the image segmentation and denoising approaches score higher recall than precision while the I2I-translation model does the opposite. Furthermore, $\text{DAE}^{(\text{bin})}_{\lambda=10}$ still creates square-like artifacts as shown in the prediction sample 5.2c.

The image segmentation approaches outperforms the baseline considering all metrics, while the $\text{DAE}^{(\text{bin})}_{\lambda=10}$ and $\text{cGAN}_{\lambda=50}$ performs equal to and worse than the reference scores, respectively. We observe that the $\text{UNET}^{(\text{bin})}_{\lambda=100}$ achieves overall good results as well as the highest F_1-score and distance RMSE. The $\text{UNET}^{(\text{bin})}_{\lambda=50}$ model has a recall that is 6 percentage points higher than the $\text{UNET}^{(\text{bin})}_{\lambda=100}$ model, but falls short on the precision metric. Thus, the F_1-score is in favor of $\text{UNET}^{(\text{bin})}_{\lambda=100}$.



(a) Sparse arc USS input. **(b)** Sparse Lidar GT. **(c)** $\text{DAE}^{(\text{bin})}_{\lambda=10}$ prediction.

(d) $\text{UNET}^{(\text{bin})}_{\lambda=50}$ prediction. **(e)** $\text{UNET}^{(\text{bin})}_{\lambda=100}$ prediction. **(f)** $\text{cGAN}_{\lambda=50}$ prediction.

Figure 5.2: Predictions from models trained on $\mathcal{D}^{(\text{sparse})}$, namely **(c)** $\text{DAE}^{(\text{bin})}_{\lambda=10}$, **(d)** $\text{UNET}^{(\text{bin})}_{\lambda=50}$, **(e)** $\text{UNET}^{(\text{bin})}_{\lambda=100}$ and **(f)** $\text{cGAN}_{\lambda=50}$. We can also see **(a)** the input and corresponding **(b)** ground truth frames. The grids are collected between 09:26:07 and 09:27:07 at 2023-04-05. Furthermore, yellow, red and black denotes occupied, unknown and free-space, respectively.

Table 5.3: Evaluation results from the trained models on $\mathcal{D}^{(\text{sparse})}$ as well as a baseline comparison.

Model	Precision [%]	Recall [%]	F_1-score [%]	Accuracy [%]	Distance RMSE [m]
None	5.33	38.46	9.36	93.81	2.5024
$\text{DAE}^{(\text{bin})}_{\lambda=10}$	5.39	41.10	9.53	93.51	2.3026
$\text{UNET}^{(\text{bin})}_{\lambda=50}$	7.42	**46.86**	12.81	94.70	1.8206
$\text{UNET}^{(\text{bin})}_{\lambda=100}$	9.63	40.40	**15.56**	96.36	**1.7277**
$\text{cGAN}_{\lambda=50}$	**71.51**	2.20	4.27	**99.18**	1.8383

5.2.3 Dense Data

Finally, the best performing models trained on $\mathcal{D}^{(\text{dense})}$ are $\text{DAE}^{(\text{mse})}$, $\text{UNET}^{(\text{ce})}$, $\text{cGAN}_{\lambda=10}$ and $\text{cGAN}_{\lambda=100}$. The evaluation metrics and prediction samples from the models are shown in Table 5.4 and Figure 5.3, respectively. We observe that the cGAN performs both worse and better than the baseline considering the F_1-score and distance RMSE, respecitively. The autoencoder using the MSE loss performs poorly and tend to predict mainly free space and unknown values, as shown in the low recall and high accuracy score. This is probably caused by the $\text{DAE}^{(\text{mse})}$ not considering the class imbalance as well as the MSE objective averaging out the low occupied values (0), as seen in Figure 5.3c.

In terms of the F_1-score, the $\text{UNET}^{(\text{ce})}$ correctly classifies the largest amount of occupied pixels compared to the other models. Interestingly, the $\text{cGAN}_{\lambda=10}$ model has a better distance RMSE, but considerably worse F_1-score. This raises the question of which model is the more suitable and is further discussed in Section 5.3.1. Moreover, we observe that the U-Net outperforms the baseline on basically all accounts. As a final comment, the U-Net yet again scores best in regards to the recall and F_1-score, with corresponding results for the cGAN regarding the precision and accuracy metrics.

(a) Dense arc USS input.

(b) Dense Lidar GT.

(c) DAE$^{(\text{mse})}$ prediction.

(d) UNET$^{(\text{ce})}$ prediction.

(e) cGAN$_{\lambda=10}$ prediction.

(f) cGAN$_{\lambda=100}$ prediction.

Figure 5.3: Predictions from models trained on $\mathcal{D}^{(\text{dense})}$, namely **(c)** DAE$^{(\text{mse})}$, **(d)** UNET$^{(\text{ce})}$, **(e)** cGAN$_{\lambda=10}$ and **(f)** cGAN$_{\lambda=100}$. We can also see **(a)** the input and corresponding **(b)** ground truth frames. The grids are collected between 08:12:45 and 08:13:45 at 2023-04-05. Furthermore, yellow, red and black denotes occupied, unknown and free-space, respectively.

Table 5.4: Evaluation results from the trained models on $\mathcal{D}^{(\text{dense})}$ as well as a baseline comparison.

Model	Precision [%]	Recall [%]	F_1-score [%]	Accuracy [%]	Distance RMSE [m]
None	5.33	38.46	9.36	93.81	2.5024
DAE$^{(\text{mse})}$	0.57	0.01	0.02	**99.00**	1.9224
UNET$^{(\text{ce})}$	6.44	**51.54**	**11.45**	92.15	1.9346
cGAN$_{\lambda=10}$	13.29	4.40	6.61	98.77	**1.8059**
cGAN$_{\lambda=100}$	**34.62**	2.14	4.03	**99.00**	1.8631

5.3 Discussion

The evaluation of the models show that different approaches perform both better and worse than the baseline comparisons. In this section we discuss interesting aspects

and findings of the thesis results as well as comment our approach and possible shortcomings. First we discuss the evaluation framework and assess its reliability.

5.3.1 Evaluation Framework

The evaluation framework uses two different evaluation approaches: the first evaluates the models in terms of a pixelwise classification problem and the second in terms of virtual distance estimations. Combining both the gridwise F_1-score and accuracy with the distance RMSE gives a good overview of the models performance, however no framework is ever perfect. In the current implementation of the distance metric, all linear search angles considered, even the ones that do not encounter any occupancies in both the prediction and ground truth frame. Such redundant angle searches can be disregarded in an effort to make the metric more focused on showing errors only when an occupancy is present. The current implementation can lead to contradictions between the F_1-score and distance RMSE. For example, we see in Section 5.2.3 that UNET$^{(\text{ce})}$ outperforms cGAN$_{\lambda=10}$ on the F_1 metric, but the opposite is true for the distance RMSE. Another example of contradictory scores is seen in Table 5.2, where the cGAN$_{\lambda=10}$ again performs poorly on the F_1-score, but well on the distance RMSE metric. A more occupancy focused implementation of the distance based evaluation could show a more streamlined result with the F_1-score. However, the two scores are based on class versus distance evaluation, which raises the question if either approach is better than the other.

An alternative to the pixelwise classification measurements is to base the metrics on virtual distances instead, rather than grid labels. This evaluation scheme is better suited for the objective of LSP, namely to as precisely as possible detect obstacle positions. The current classification scores focuses more on predicting exact pixels instead of capturing the general object shape instead. However, the aim of the thesis explicitly states that we are to devise methods to mimic Lidar generated occupancy grids, which makes current gridwise framework acceptable for our purposes. In conclusion, basing the classification metrics on distances would most certainly make the F_1-score and distance RMSE more connected. A concrete proposal for the distance based classification metrics are presented in Section 6.1.

5.3.2 Data Representations and Preprocessing

We see that evaluation scores differ quite a lot when comparing models trained on the different data sets. For example, UNET$^{(\text{bin})}_{\lambda=10}$ and DAE$^{(\text{bin})}_{\lambda=10}$ gets a performance gain in the F_1-score, accuracy and distance RMSE metrics when trained on $\mathcal{D}^{(\text{sparse})}$ compared to $\mathcal{D}^{(\text{basic})}$. The same can be said for cGAN$_{\lambda=10}$, which improves its F_1-score and distance RMSE when trained on $\mathcal{D}^{(\text{dense})}$ compared to the other two data sets. See Tables A.1, A.2 and A.3 for the results. Thus, the input data representation plays a major role in model performance. It is trivial to realise that if the input and ground truth grids look more alike, a NN will perform the mapping task more easily. Furthermore, the chosen representations correspond to the standard way ultrasonic signals are modeled, as presented in Section 2.3, and extensive research would be required to get more Lidar-like grids. This concludes that the used USS

representations were well suited for this thesis, considering the narrow project time frame.

However, it is still of interest to investigate new representation schemes, as the results suggest better performance for different grid types. For example, increase the number of input data dimensions, as this usually makes neural networks more robust against overfitting. The input could therefore potentially be changed to a multidimensional tensor containing several USS embeddings, such as using all representations shown in Figure 3.6, or a time series of frames. Corresponding models could then be adapted to perform 3D convolutions for better information processing. Another plausible input scheme would be to incorporate more relevant meta data, e.g. as an additional vector, to increase the number of dimensions.

It is important to note that only one sample was drawn from the cGAN models when evaluating the I2I translation approach. Several samples could have instead been generated with the model and then averaged over, since the cGAN draws samples from a learnt distribution. The variance of the created distribution would also be interesting to consider, to investigate how much the outputs differ given the same USS input.

Another aspect that influences the results are the performed preprocessing steps. Three main data refinements were implemented, namely:

1. Removing frame pairs with no USS nor Lidar occupancy's present.

2. Setting a speed limit and time difference threshold to avoid frame pairs of different scenarios to be matched.

3. Masking Lidar grids to only cover the USS effective distances.

Two additional preprocessing steps were considered, however they were not implemented due to time limitations. The first is subsampling of frame pairs, motivated by the fact that some grids can be almost identical to the neighbouring samples in a training sequence. This makes the networks more prone to overfitting. Implementing a carefully designed subsampling process will not only help the predictions to become more robust, but the training time will also decrease as there exists less training samples. The second is demi-denoising, which aims to remove false USS activations caused by background noise. The aim is to simply remove all USS activations in either the upper or lower half of the grid if no corresponding occupancy is detected by the Lidar.

5.3.3 Problem Approach

Three approaches were used to map USS frames into Lidar grids, namely image denoising, image segmentation and image-to-image translation. We start with assessing the image denoising approach based on the results. Neither of the proposed DAE models performed very well considering the F_1-score and distance RMSE for any data set and was always outperformed. At best, $\text{DAE}^{(\text{bin})}_{\lambda=10}$ could perform equally

well as the baseline for $\mathcal{D}^{(\text{sparse})}$. Thus, it is questionable if the used autoencoder architecture has the capacity of learning the desired domain mapping. The implemented NN architecture has few layers compared to e.g. the U-Net and additional layers could be added for more trainable parameters. However, additional layers could lead to vanishing gradient problems, as discussed in Section 2.1.1.1. These small gradients cannot be dealt with by adding skip connections, since this compromises the AE architecture and prevents the encoder to create input representations in the latent layer. We conclude that seeing the problem as a denoising task is not viable.

Regarding any GAN model, training convergence is often a fleeting state. In the ideal case the model should reach a point after which the generator successfully fools the discriminator, and through this reveal that the training has converged. The results show that we do not find such a point for the cGAN on any of the data sets.

It is difficult to pinpoint the exact reason for this, but it seems that for $\mathcal{D}^{(\text{basic})}$ and $\mathcal{D}^{(\text{sparse})}$, the generator finds that a very sparse output fools the discriminator most of the time. This is reminiscent of the same symptoms as mode collapse, however both networks keep on learning. One reason for this could be that the loss function is not very well suited to this specific task. The L1-term measures the similarity between a generated grid and the ground truth, but since the input USS grid and ground truth are still very similar in most regards, this measurement is very rough. This also affects the BCE-term as parts that remain static between the grids are still very likely to be judged as real by the discriminator over the entire training. While the results on $\mathcal{D}^{(\text{dense})}$ show that this data representation seems to be more suited to the cGAN, it suffers the same problems.

Applying an image segmentation approach to mimic the Lidar frames was proven to be more successful than the other two approaches in terms of getting consistently good results. The implemented U-Net architecture tends to get higher recall than precision score, meaning that the model captures larger amounts of occupied pixels of the cost of making many inaccurate predictions. $\text{UNET}^{(\text{bin})}_{\lambda=100}$ managed to achieve the lowest distance RMSE and highest F_1-score with a decent accuracy of 96 %, when trained on $\mathcal{D}^{(\text{sparse})}$. Thus, we have proven that it is possible to mimic Lidar grids with neural networks with improved quality compared to the input representations. However, the best result is quite unreliable as $F_1 < 0.5$ and the distance RMSE is above 1.5 meters in estimated variance.

The $\text{UNET}^{(\text{bin})}_{\lambda=100}$ result points towards that something drastic needs to change to improve the results and that none of the test approaches performs well enough to be used in LSP. We elaborated in Section 5.3.1 that distances are more important than pixel values for the LSP objective. Thus, it would be interesting to use a custom distance based loss function to train a network. Maybe even in combination with the image processing based losses.

6

Conclusion

The purpose of this thesis has been to explore the possibility to enhance USS perception by using Lidar data as ground truth. We conclude that the performance of all models in doing this is highly dependent on the data representation. Using the U-Net in a hybrid image segmentation/translation setting with a loss function representing both parts shows promise in performing this task however, as it outperforms the baseline on two of the three data sets. On the other hand, we discover that simple image denoising models do not seem to be sufficient to enhance USS perception. Some results of the I2I cGAN show improvements, but more research is needed to determine if image-to-image translation is a viable approach.

We have explored three different deep learning approaches resulting in five different models: $\text{DAE}^{(\text{bin})}$, $\text{DAE}^{(\text{mse})}$, $\text{UNET}^{(\text{bin})}$, $\text{UNET}^{(\text{ce})}$ and I2I cGAN. In order to train these models, we utilized data provided by VCC to generate three different data sets containing pairs of USS and Lidar occupancy grids, $\mathcal{D}^{(\text{basic})}$, $\mathcal{D}^{(\text{sparse})}$ and $\mathcal{D}^{(\text{dense})}$. In addition, we developed an evaluation framework in order to compare the performance of these networks. This framework is includes a set of metrics that reflect the accuracy of the networks in representing the real-world environment.

The U-Net models trained on $\mathcal{D}^{(\text{sparse})}$ and $\mathcal{D}^{(\text{dense})}$, sport a significant improvement in F_1-score and RMSE-distance. The $\text{DAE}^{(\text{bin})}_{\lambda=10}$ trained on $\mathcal{D}^{(\text{sparse})}$ outperforms the baseline marginally on each metric, but fails to do the same on $\mathcal{D}^{(\text{basic})}$. The alternate model $\text{DAE}^{(\text{mse})}$, adapted for dense data, generally fails to capture occupied cells and as a result also fails to outperform the baseline.

The poor results of the cGAN on both $\mathcal{D}^{(\text{basic})}$ and $\mathcal{D}^{(\text{sparse})}$ show that the model is unable to learn a useful mapping to the ground truth distribution, most likely due to the similarity between input and ground truth data in uninteresting regions affecting the GAN training scheme. On $\mathcal{D}^{(\text{dense})}$ the cGAN succeeds in reproducing relevant features of the ground truth grid. The regions of occupied cells in the output are somewhat shifted in comparison to the ground truth however. This affects the evaluation, as the classification-based metrics calculations are binary, making it hard to draw any final conclusions about its performance.

6.1 Future Work

Based on the results and the discussion, we believe that new approaches and alternative data representations can be used to improve the quality of the predictions. Three key areas have been identified where more work can be done, namely data, evaluation and approach.

6.1.1 Data Representations and Preprocessing

While our created data representations showed promise, we still do not utilize all aspects of the ultrasonic data, such as the signal strength. One avenue open to exploration is therefore to provide all information contained in the signalways to our models, including metadata such as whether the signalway is a direct or cross echo. Another avenue is to try to remedy the fact that the current USS grids contain less information than the corresponding Lidar grids, by mapping multiple representations of a USS grid to a Lidar grid. Finally, we chose not to explore time series models due to the fact that the initial data contained a non-negligible amount of specific test cases, and we deemed that the networks had a chance to overfit to these. The final data was gathered in real driver scenarios and was thus more general, opening up the possibility to leverage time series models for this task.

The current data sets contain instances where a range of USS grids mapped to Lidar grids are very similar, for instance if the car is driving very slowly. This means we are essentially creating duplicates of our data. We propose that any future work makes use of subsampling in these situations to make sure that any data set is properly balanced.

In addition, we have not explored the level of noise in our data. One example of a situation that would generate a noisy data point would be a faulty Lidar sensor causing detections at the rear bumper due to dirt while both types of sensors detect a real object at the front bumper. A solution to this and associated problems would be to filter data where detections in input and ground truth data are not in the same region.

6.1.2 Evaluation Framework

What we are really interested in when leveraging these models is reducing is the distance between occupied cells in the output and ground truth grid. It would be of great value to develop and test a new loss function that reflects this distance explicitly. In connection to this, an improvement in the evaluation framework that has yet to be included is to only consider detection distances in the distance error measurement, rather than the grid as a whole. Another improvement would be to have an error tolerance in the tallying of the confusion matrix elements used for the other performance metrics. Prediction labels can be created by using the virtual distance measuring algorithm proposed in Section 4.1.2 and would redefine the classes as following;

1. **True Positive (TP)** - A predicted occupancy distance falls within an acceptance tolerance from the corresponding ground truth value.

2. **False Positive (FP)** - The predicted occupancy distance falls outside the acceptance interval created by the ground truth. Or the ground truth detects no occupancy, but an occupancy distance is predicted prior to the maximal search range.

3. **True Negative (TN)** - Both GT and prediction indicates the max range value.

4. **False Negative (FN)** - The prediction indicates no occupancy, but the GT distance is smaller than the maximal search range.

6.1.3 Problem Approach

There is still room for improvements in our models in the tuning of hyperparameters. Examples of parameters that have not been optimized for are learning parameters, a fine-grained search over the L1-weighting parameter λ, and weighting schemes. While the current weighting schemes are motivated by imbalance, it would be interesting to see if increased weights for occupied pixels would improve the performance for the sparse models, as occupied cells are of greater importance to this task.

Ultimately, our implementation of the I2I cGAN is designed to map an image in a domain that is essentially the same as the target domain in regards to region and shape, while changing characteristics such as texture. New work in the domain of image-to-image translation, such as DiscoGANs, has shown the possibility to map between different domains, changing the shape of objects, such as handbags to shoes, while preserving texture. This seems to be a much more fitting model in this context, and any future work is encouraged to explore this further. It would also be of great interest to explore the combination of WGANs with I2I cGANs in this use case to see if this would alleviate the problem of mode collapse.

Bibliography

[1] Gary King and Langche Zeng. "Logistic Regression in Rare Events Data". In: *Political Analysis* 9.2 (2001), pp. 137–163.

[2] Alan B. Coppens Lawrence E. Kinsler Austin R. Frey and James V. Sanders. *Fundamentals of Acoustics*. John Wiley & Sons, 2009, pp. 127–130.

[3] Pascal Vincent et al. "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion". In: *Journal of Machine Learning Research* 11.110 (2010), pp. 3371–3408.

[4] Dale Ensminger and Leonard J. Bond. *Ultrasonics: Fundamentals, Technologies and Applications*. CRC Press and Taylor & Francis Group, 2011, pp. 20–22.

[5] Ian Goodfellow et al. "Generative Adversarial Networks". In: *Advances in Neural Information Processing Systems* 3 (2014).

[6] Mehdi Mirza and Simon Osindero. "Conditional Generative Adversarial Nets". In: *CoRR* (2014). arXiv: 1411.1784.

[7] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* (2015). arXiv: 1502.03167.

[8] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015.

[9] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Ed. by Nassir Navab et al. Springer International Publishing, 2015, pp. 234–241.

[10] Phillip Isola et al. "Image-to-Image Translation with Conditional Adversarial Networks". In: *CoRR* (2016). arXiv: 1611.07004.

[11] Alec Radford, Luke Metz, and Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2016. arXiv: 1511.06434.

[12] Martin Arjovsky and Leon Bottou. "Towards Principled Methods for Training Generative Adversarial Networks". In: *International Conference on Learning Representations*. 2017.

[13] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein Generative Adversarial Networks". In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. JMLR.org, 2017, pp. 214–223.

[14] Ian Goodfellow. *NIPS 2016 Tutorial: Generative Adversarial Networks*. 2017. arXiv: 1701.00160.

[15] Taeksoo Kim et al. "Learning to Discover Cross-Domain Relations with Generative Adversarial Networks". In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. JMLR.org, 2017, pp. 1857–1865.

[16] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980.

[17] Yuhuai Wu et al. "On the Quantitative Analysis of Decoder-Based Generative Models". In: *International Conference on Learning Representations*. 2017.

[18] Jun-Yan Zhu et al. "Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2242–2251.

[19] Ebubekir BUBER and Banu DIRI. "Performance Analysis and CPU vs GPU Comparison for Deep Learning". In: *2018 6th International Conference on Control Engineering & Information Technology (CEIT)*. 2018, pp. 1–6.

[20] Y. Choi et al. "StarGAN: Unified Generative Adversarial Networks for Multi-domain Image-to-Image Translation". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, 2018, pp. 8789–8797.

[21] Vincent Dumoulin and Francesco Visin. *A guide to convolution arithmetic for deep learning*. 2018. arXiv: 1603.07285.

[22] Phillip Isola et al. *Image-to-Image Translation with Conditional Adversarial Networks*. 2018. arXiv: 1611.07004.

[23] Noah Makow. *Wasserstein GANs for Image-to-Image Translation*. 2018.

[24] Wenming Yang et al. "Deep Learning for Single Image Super-Resolution: A Brief Review". In: *CoRR* (2018). arXiv: 1808.03344.

[25] Bernhard Mehlig. *Machine Learning with Neural Networks*. Cambridge University Press, 2021.

[26] Iqbal H. Sarker. "Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions". In: *SN Computer Science* (2021).

[27] Dominik Müller, Iñaki Soto-Rey, and Frank Kramer. "Towards a guideline for evaluation metrics in medical image segmentation". In: *BMC Research Notes* 15.1 (2022), p. 210.

[28] Yingxue Pang et al. "Image-to-Image Translation: Methods and Applications". In: *IEEE Transactions on Multimedia* 24 (2022), pp. 3859–3881.

[29] Volvo Car Corporation. *Safety*. https://www.volvocars.com/intl/v/safety/culture-vision. Accessed: 2023-05-25. 2023.

[30] Wikipedia. *Lidar*. https://en.wikipedia.org/wiki/Lidar. Accessed: 2023-01-27. 2023.

[31] Technical Committee ISO/TC 22. *ISO 8855:2011(en) Road vehicles*. URL: https://www.iso.org/obp/ui/#iso:std:iso:8855:ed-2:v1:en. (accessed: 24-04-2023).

[32] TensorFlow Community. *Intro to Autoencoders*. URL: https://www.tensorflow.org/tutorials/generative/autoencoder. (accessed: 01-04-2023).

[33] The HDF Group. *The HDF5® Library & File Format*. URL: https://www.hdfgroup.org/solutions/hdf5/. (accessed: 19-04-2023).

[34] The Luigi Authors Revision. *Luigi*. URL: https://luigi.readthedocs.io/en/stable/. (accessed: 23-04-2023).

[35] Wikipedia. *Generative model*. URL: https://en.wikipedia.org/wiki/Generative_model. (accessed: 20-04-2023).

A

Appendix A

Table A.1: All evaluation results from the trained models on $\mathcal{D}^{(\text{basic})}$.

Model	Precision [%]	Recall [%]	F_1-score [%]	Accuracy [%]	Distance RMSE [m]
None	10.04	0.91	1.67	99.11	1.8428
$\text{DAE}^{(\text{bin})}_{\lambda=10}$	3.36	68.45	6.41	83.39	2.9694
$\text{UNET}^{(\text{bin})}_{\lambda=10}$	4.90	71.46	9.16	88.23	2.1974
$\text{cGAN}_{\lambda=10}$	21.76	0.02	0.04	99.17	1.8569

Table A.2: All evaluation results from the trained models on $\mathcal{D}^{(\text{sparse})}$.

Model	Precision [%]	Recall [%]	F_1-score [%]	Accuracy [%]	Distance RMSE [m]
None	5.33	38.46	9.36	93.81	2.5024
$\text{DAE}^{(\text{bin})}_{\lambda=10}$	5.39	41.10	9.53	93.51	2.3026
$\text{UNET}^{(\text{bin})}_{\lambda=0}$	8.80	43.77	14.65	95.77	1.7713
$\text{UNET}^{(\text{bin})}_{\lambda=10}$	7.83	46.19	13.38	95.03	1.7903
$\text{UNET}^{(\text{bin})}_{\lambda=50}$	7.42	46.86	12.81	94.70	1.8206
$\text{UNET}^{(\text{bin})}_{\lambda=100}$	9.63	40.40	15.56	96.36	1.7277
$\text{UNET}^{(\text{bin})}_{1}$	7.69	45.85	13.18	94.98	1.7990
$\text{UNET}^{(\text{bin})}_{2}$	8.51	42.08	14.15	95.76	1.7730
$\text{cGAN}_{\lambda=10}$	15.01	0.02	0.03	99.17	1.8588
$\text{cGAN}_{\lambda=50}$	71.51	2.20	4.27	99.18	1.8383
$\text{cGAN}_{\lambda=100}$	50.49	1.20	2.34	99.17	1.8435

[1] $\lambda = 100$, $m_b = 16$

[2] $\lambda = 100$, $m_b = 16$, $\alpha = 0.0001$

Table A.3: All evaluation results from the trained models on $\mathcal{D}^{(\text{dense})}$.

Model	Precision [%]	Recall [%]	F_1-score [%]	Accuracy [%]	Distance RMSE [m]
None	5.33	38.46	9.36	93.81	2.5024
$\text{DAE}^{(\text{mse})}$	0.57	0.01	0.02	99.00	1.9224
$\text{UNET}^{(\text{ce})}$	6.44	51.54	11.45	92.15	1.9346
$\text{cGAN}_{\lambda=10}$	13.29	4.40	6.61	98.77	1.8059
$\text{cGAN}_{\lambda=50}$	26.52	1.40	2.67	98.99	1.8739
$\text{cGAN}_{\lambda=100}$	34.62	2.14	4.03	99.00	1.8631

B

Appendix B

B.1 Neural Network Architectures

B.1.1 Autoencoder

The network takes a 192×128 OG as input and output the same dimension. All layers uses *same*-style option pads the layer input such that the ith dimension in the hidden layer becomes

$$d_i = \left\lceil \frac{\dim(\boldsymbol{x}, i)}{s_i} \right\rceil,$$

where s_i is the corresponding stride and $\dim(\boldsymbol{x}, i)$ is the number of dimension along the ith axis. See Table B.1 for more layer information.

Table B.1: Autoencoder architecture.

Layer	Filters	Kernel size	Stride	Activation
Conv2D	16	3×3	2×2	ReLU
Conv2D	8	3×3	2×2	ReLU
Conv2D	4	3×3	2×2	ReLU
TConv2D	4	3×3	2×2	ReLU
TConv2D	8	3×3	2×2	ReLU
TConv2D	16	3×3	2×2	ReLU
Conv2D	1	3×3	1×1	Sigmoid

B.1.2 U-Net

All U-Net layers uses *same* style padding, a 4×4 kernel size. Furthermore, Leaky ReLU (LReLU) is a modified ReLU function, which allows a small gradient for inactive units, namely

$$\text{LeakyReLU}(x) = \begin{cases} x, & \text{if } x > 0 \\ \eta x, & \text{otherwise,} \end{cases}$$

with $\eta = 0.3$. The dropout scheme drops a fraction $q = 0.5$ of the neurons randomly. Depending if a binary or multilabel output is required are final layer (1) respectively

(2) used. See Table B.2 for more layer information.

Table B.2: U-Net architecture. The skip label indicate which layers that are connected

Layer	Filters	Stride	Activation	BatchNorm	Dropout	Skip
Conv2D	32	2×2	LReLU	-	-	A
Conv2D	64	2×2	LReLU	✓	-	B
Conv2D	128	2×2	LReLU	✓	-	C
Conv2D	256	2×2	LReLU	✓	-	D
Conv2D	512	2×2	LReLU	✓	-	E
Conv2D	512	2×2	LReLU	✓	-	F
Conv2D	512	3×2	LReLU	✓	-	-
TConv2D	512	3×2	LReLU	✓	✓	F
TConv2D	512	2×2	LReLU	✓	✓	E
TConv2D	256	2×2	LReLU	✓	✓	D
TConv2D	128	2×2	LReLU	✓	-	C
TConv2D	64	2×2	LReLU	✓	-	B
TConv2D	32	2×2	LReLU	✓	-	A
(1)TConv2D	1	2×2	Sigmoid	-	-	-
(2)TConv2D	3	2×2	-	-	-	-

B.1.3 Image-to-image cGAN

The U-Net core used for the I2I cGAN is the same as the one described in Section B.1.2 above, but with an initial Gaussian noise layer of shape $(192 \times 128 \times 1)$, representing the prior distribution $p_{\mathbf{z}}(\mathbf{z})$. The Gaussian noise applied is zero-mean with a standard deviation of 0.1, as motivated in Section 4.4.

The discriminator is in essence a CNN with two input layers of shape $(192 \times 128 \times 1)$. In the original implementation of this PatchGAN, the authors used input images of size $(256 \times 256 \times 1)$, and as such produces symmetrical patches of size (30×30) [22]. As the dimensions used for our grids are decided by the domain-specific problem, we elected instead to use rectangular patches, as resizing the output to a square format would reduce the information captured along one dimension. This yields an output dimension of $(22 \times 14 \times 1)$.

The zero padding scheme adds columns and rows of zeros on all sides of an image tensor. A kernel size of 4×4 is used for all layers. See Table B.3 for more layer information.

Table B.3: CNN architecture of the cGAN discriminator.

Layer	Filters	Stride	Activation	BatchNorm	Padding
Conv2D	64	2×2	LReLU	-	Same
Conv2D	128	2×2	LReLU	✓	Same
Conv2D	256	2×2	LReLU	✓	Same
Conv2D	512	1×1	LReLU	✓	Zero
Conv2D	1	1×1	Sigmoid	-	Zero

This structure is intended to extract and learn the features necessary in order to determine the probability that patches of overlapping convolutions over an input grid represent a true Lidar grid. The the design of this architecture by Isola et al. [22] was influenced by Radford et al. [11], which motivate the use of e.g. strided convolutions instead of pooling layers, batch normalization, removing fully connected hidden layers and Leaky ReLU activation.

C

Appendix C

C.1 Data Preprocessing

Both the ultrasonic sensor and Lidar data need to be processed in order to convert the data streams into frames with useful information. In this section, this process is described in detail. Additionally, a merging scheme is developed to secure that training and ground truth data is matched in a way that reflects reality.

C.1.1 Ultrasonic Sensor Pipeline

Due to bandwidth limitations, the data pertaining to a signalway is divided into data packets when received. These packets contain metadata such as signalway ID, timestamp, etc. as well as the two echoes each with distance, timestamp, amplitude and time of flight amongst others. The information used for constructing the training data is distance, signalway ID, direction and location of sensors and the timestamp. The packets are gathered into logs of minute length and all logs pertaining to a test drive constitutes a data stream.

In order to convert the data streams into the desired 2D occupancy grid format data, processing is required. This is done by implementation of a Luigi pipeline which in essence takes the paths to HDF5 log files as input, and outputs new HDF5 files containing frames of shape 180×128. Luigi pipelines consist primarily of tasks, and is where all processing and computations are performed [34].

The USS preprocessing pipeline consists of five separate preprocessing tasks and a log streaming task, illustrated in Figure C.1. Notice that the structure is that of a directed acyclic graph (DAG), which is a requirement of the Luigi framework in order to handle parallelization of tasks.

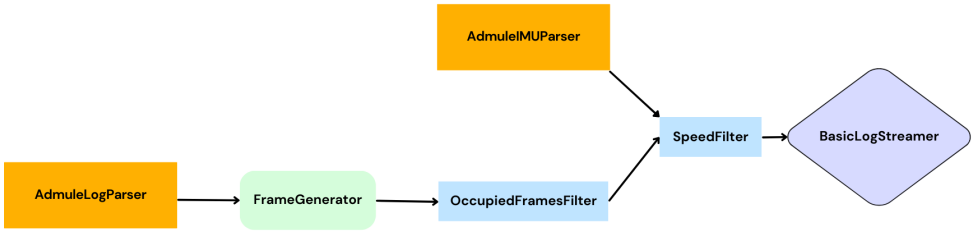


Figure C.1: Graph depicting the workflow of the USS preprocessing pipeline.

A more detailed description of the tasks is given below:

1. The first task is to stitch the log data packets containing the fragmented signalway information into one logical grouping of signalway data fields: signalway ID, timestamp for each data packet, sensor blind distance, sensor max detection distance, echo distances and signalway status (if direct or indirect signalway) being of greatest interest. Due to redundant measurements being made in order not to lose measurement data, duplicate signalways exist. These are removed before proceeding to the next task.

2. (a) Frames are generated by grouping signalways into firing sequences which are then pairwise added, one after the other: $1 + 2$, $2 + 3$, etc. Thus, sequence 1 and 2 constitutes one frame, sequence 2 and 3 another. For each signalway, the corresponding transmitting sensor ID is then extracted.

   (b) Using the position and angle of the sensor and the echo distance, appropriate scaling and translating converts the coordinate of a detection to the coordinate system displayed in Figure 3.3. The kind of occupancy representation generated at this coordinate depends on configuration passed in the streaming task (6). An occupancy representation is only generated if the echo distance is greater than the blind distance and lesser than the max detection distance of the sensor.

   (c) Each frame is paired with a timestamp which is the average of the data packet timestamps, $T_{\text{packets}} = \frac{1}{n}\sum_{i=1}^{n} T_{\text{packet},i}$, where n is the number of packets, and the average $T = \frac{1}{m}\sum_{i}^{m} T_{\text{packets},i}$ over all m signalways that comprise the frame.

3. Frames that do not contain any detections are removed, since they provide no relevant information for the models to learn from.

4. Intertial Measurement Unit (IMU) data is parsed from IMU logs, where the vehicle speed is of interest.

5. The vehicle speed from the previous task is used to filter out speeds below a predefined maximum value v_{max}. This is done because the frame timestamps are not exact, and do not exactly match up with the timestamps for the ground truth frames, and this time difference introduces an error in the location of an occupied cell, which is proportional to the speed of the vehicle. More details regarding this maximum value and the corresponding error threshold is described in Section C.1.4.

6. The last task is a streaming task that is placed in front of the graph since Luigi resolves the dependencies of each task in a backwards fashion. This task creates one dependency chain for each log file to be read with the rest in parallel manner. Here, configuration such as what kind of data set should be created is passed to the rest of the pipeline. The datasets created using this pipeline are described and shown in Section 3.2.

Adding to the description of the frame generation task, there is no overlap in signal ways between any two consecutive firing sequences, thus one can create a complete sensor perception image (and a resulting frame) from combining firing sequences either as $1 + 2$, $3 + 4$ etc., or $1 + 2$, $2 + 3$ etc. Here, the latter option was chosen as this yields a time difference between frames that is 33 percent shorter than the first option, resulting in more frames.

C.1.2 Lidar Pipeline

Each Lidar HDF5 file described in Section 3.1.2 needs to be converted into occupancy grids. Thus, a preprocessing pipeline is created, which filters and project the three dimensional measurements into a two dimensional grid as well as performing a masking operation. Initially are all sensor rotations merged for each timestamp, which means that a rotation now contains measurements across the entire environment around the ego-vehicle. A threshold z_{ground} is needed in order to decide if an entry is a ground reflection or obstacle detection. All coordinates with $z \geq z_{\text{ground}} = 0.25$ meters is marked as occupied and otherwise free space. Furthermore, the Lidar sensors can capture artifacts above the vehicle, thus is a second threshold $z_{\text{max}} = 2.5$ meters introduced. Measurements with $z \geq z_{\text{max}}$ are therefore ignored. Filtering also needs to be performed on the x and y coordinates. Coordinates that satisfy

$$\begin{aligned} x_{\text{min}} < & x < x_{\text{max}} \\ |y| < & y_{\text{max}} \end{aligned}$$

is kept, where $x_{\text{min}} = -7.4$ meters, $x_{\text{max}} = 10.6$ meters and $y_{\text{max}} = 6.4$ meters. The remaining points in the rotation are mapped to a 180×128 grid coordinates. In order to reduce noise, must at least 4 measurements indicate a pixel as occupied for

it to be marked as such. The dense algorithm grid is initially all pixels labeled as unknown, and free space measurements flips pixels in a small radius to free space and thus indicating what parts of the map that has been measured. Figure 3.7a shows example outputs from the dense and sparse algorithms.

Because the short range Lidar range exceeds the USS range, a mask is put around the ego vehicle in order to constrain the information to objects that both sensor types can observe. A rectangular mask is also put over the vehicle origin, since this area is not of interest to make predictions upon. Furthermore, the Lidar can experience light reflections from the vehicle, which is also masked out by making the polygon a bit larger than the actual vehicle grid representation. Figure 3.9 shows how the the frames shown in Figure 3.7 looks after masking.

C.1.3 Merging USS and Lidar Frames

When merging the USS grids with corresponding Lidar grids there are three essential steps that are performed; match the corresponding files, match the timestamps and remove frame pairs where the Lidar frame has no detections. Firstly are the files matched based on a common file ID and sequence ID, which are shared for USS and Lidar grid files. Stand alone files is not further processed, since the supervised models can not be trained on this data.

Secondly, a timestamp matching is performed on each file pair which matches the timestamps of USS and Lidar grids. To ensure that the two frames represent the same situation a time tolerance T_{frame} is introduced and set according to Section C.1.4. If the time difference between the best matching timestamps exceed T_{frame}, the frame in question is rejected. Thirdly, are frames removed where the matched Lidar frames contain no activation. Lidar frames without activations will only contribute to noise in training of models, since the USS frames have been filtered from dead frames.

A final preprocessing step is to pad the height of the frames by 6 pixels with value 50 or 100 depending on the data set. The reason for using a network input of size 192×128 instead of the original 180×128 occupancy grid dimensions is because $192 = 2^6 \cdot 3$ is more divisible by 2 than $180 = 2^2 \cdot 3^2 \cdot 5$. Thus, the input is easier to down and upscale and allows deeper networks. This means that each 180×128 occupancy grid is prepossessed before training by padding 3 rows of free space values at the top and bottom of the input.

C.1.4 Setting the time and speed thresholds

The time tolerance is larger for slow speeds and smaller for fast speeds. The threshold parameter can be expressed as

$$T_{\text{frame}} = \frac{d_{\text{error}}}{v_{\text{max}}}$$

where d_{error} is the allowed distance which a pixel can differ between two frames and v_{max} is the maximum speed. In Figure C.2 we see the relation between T_{frame}

and $v_{\max}$ for different distance errors. Greater speeds and time threshold values are allowed for larger accepted error and vice-versa. The accepted error $d_{\text{error}} = 0.1$ meters is set, since it corresponds to max 1 pixel difference in the grid representation which is deemed as reasonable. The max velocity is set by evaluating how much data that is lost by setting the speed limit. In Figure 3.2 the speed distribution is shown for the USS data that is to be merged. The distribution resembles an exponential decay function, meaning that the vast majority of the speeds are low, while few data points are captured at a high speed environment. It is deemed reasonable that 80 % of the USS data should pass the filter, therefore is the 80th percentile of the speed distribution calculated and used to set the time threshold. The parameters is set to $T_{\text{frame}} = 31.65$ milliseconds and $v_{\max} = 11.37$ kilometers per hour.

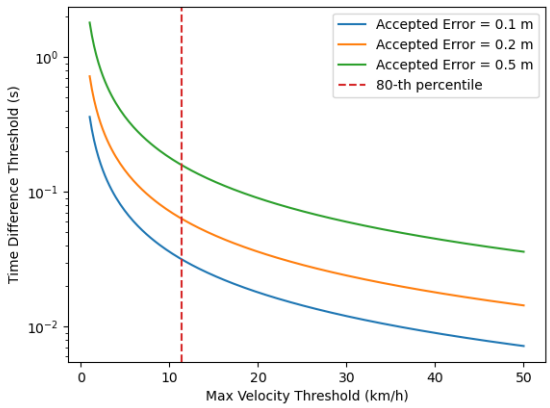


Figure C.2: The relationship between the time-pairing threshold and maximum allowed velocity for different accepted distance errors. The red, dashed line correspond to the 80th percentile of the speed distribution shown in Figure 3.2.

D

Appendix D

With optimization problems, questions regarding optimality conditions related to convergence and existence of optimal solutions arise. Goodfellow et al. provide two propositions and a theorem in order to answer these questions, while providing a caveat that these theoretical results are derived in a non-parametric setting, i.e. a model with infinite capacity and studying convergence in the space of probability density functions [5].

Furthermore, GAN's represent a limited family of p_g-distributions and optimization of $\boldsymbol{\theta}_g$, meaning that the proofs do not apply under realistic conditions [5]. The proofs are here omitted.

Firstly, fixing G gives the optimal discriminator D for a given G:

$$D^*_G(\boldsymbol{x}) = \frac{p_d(\boldsymbol{x})}{p_d(\boldsymbol{x}) + p_g(\boldsymbol{x})}$$

Secondly, reformulating the minimax game as the virtual training criterion:

$$C(G) = \max_D V(G, D) = \mathbb{E}_{\boldsymbol{x}\sim p_d}\Big[\log(D^*_G(\boldsymbol{x}))\Big] + \mathbb{E}_{\boldsymbol{x}\sim p_g}\Big[\log(1 - D^*_G(\boldsymbol{x}))\Big]$$

for a given G, where the global minimum is achieved if and only if $p_g = p_d$.

Thirdly, p_g converges to p_d if the networks representing G and D are given enough capacity, and at each step of algorithm 1, the discriminator is allowed to reach its optimum given G, and p_g is updated so as to improve the criterion $C(G)$.

Combining these three results provides the existence of a global optimum for the objective function for a given G, equal to $D^*_G(\boldsymbol{x}) = 1/2$. This optimum is achieved if and only if $p_g = p_d$, which is possible since p_g does converge to p_d under the right conditions.

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY