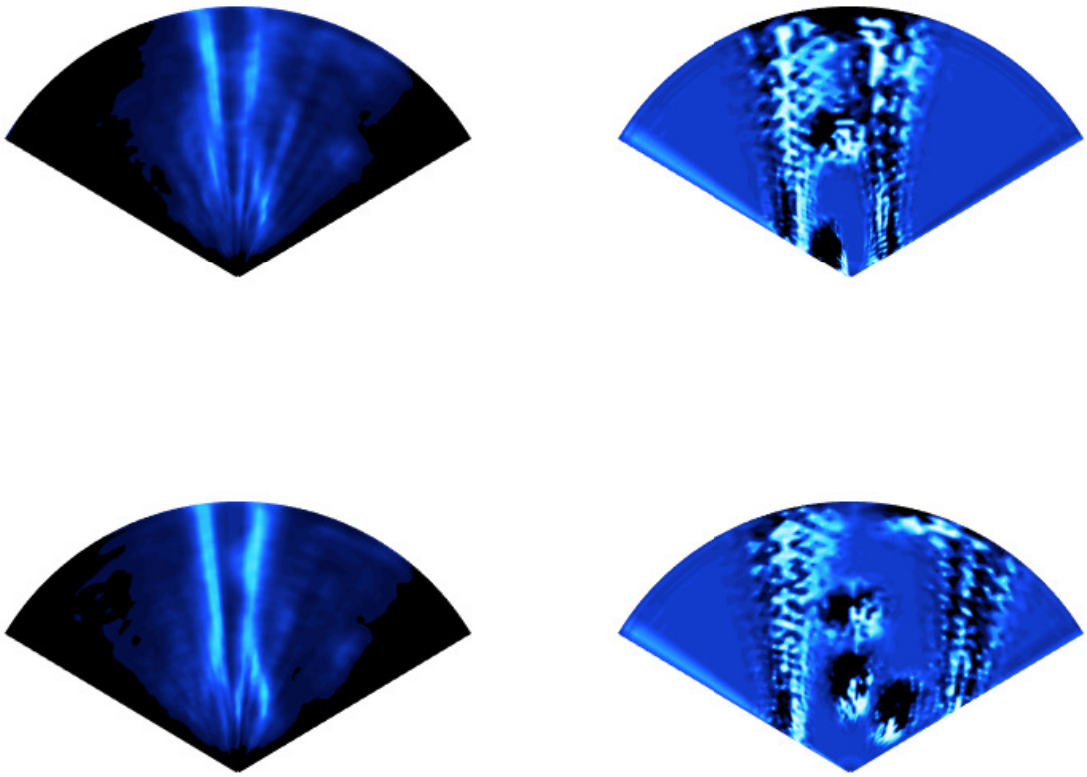
CHALMERS
UNIVERSITY OF TECHNOLOGY



Radar Sensor Modelling Using Deep Generative Networks For Verification of Autonomous Driving

Master's thesis in Complex Adaptive Systems

LOVISA HAGSTRÖM & LISA SJÖBLOM

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

Master's thesis 2019

Radar Sensor Modelling Using Deep Generative Networks For Verification of Autonomous Driving

LOVISA HAGSTRÖM & LISA SJÖBLOM

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2019

Radar Sensor Modelling Using Deep Generative Networks For Verification of Autonomous Driving
LOVISA HAGSTRÖM & LISA SJÖBLOM

Supervisor: Henik Arnelid, Zenuity
Examiner: Lars Hammarstrand, Chalmers

Master's Thesis 2019
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Example outputs from the radar models. The first column corresponds to a VAE model and the second column to a GAN model.

Typeset in LaTeX
Note that in some pdf-readers (for example *Adobe Acrobat Reader*), the figures in the thesis look strange. This is not the case when reading it in e.g. *Microsoft Edge* or *Google Chrome*. This document is best printed in color.
Gothenburg, Sweden 2019

Radar Sensor Modelling Using Deep Generative Networks For Verification of Autonomous Driving
LOVISA HAGSTRÖM & LISA SJÖBLOM
Department of Electrical Engineering
Chalmers University of Technology

Abstract

The current advancements within the field of autonomous vehicles call for the development of reliable and robust testing methods of these systems. While empirical testing is necessary, virtual verification through simulations has the advantage of being safe, fast and cost effective. However, to recreate a realistic driving scenario in a simulation environment, correct models of the sensors used by self-driving systems are needed. Some of the sensors used by these systems are the lidar and the radar sensor. The purpose of this thesis is to construct a model of a radar sensor, based on data from both a lidar and a radar sensor.

To model a radar sensor, deep generative networks such as conditional variational autoencoders (CVAE) and conditional generative adversarial networks (CGAN) are utilised. The networks are trained on data from a radar sensor, while data from a lidar is used as conditional input. The conditional input contains an environmental description from which the radar model can create its output. Both radar and lidar data are converted into suitable formats before training the models.

Both types of generative networks show potential to perform accurate radar modelling. The CVAE is able to learn general behaviour of the radar sensor, while it has difficulty adapting to specific scenarios. The CGAN, on the other hand, shows sensitivity to the conditional input, but lacks the generic properties captured by the CVAE. For both of the network results, it is apparent that errors in the data have an adverse effect on the model performance. Future improvements that can be made for the radar modelling are to include more features in the data that is given to the networks, to create a network that combines a VAE and a GAN, and to improve on the evaluation methods for the network output.

Keywords: Autonomous Driving, Safety, Radar modelling, Virtual verification, Deep Generative Networks, Variational Autoencoders, Generative Adversarial Networks

Acknowledgements

This thesis was made possible by a number of people besides the authors. First of all, we would like to thank our examiner and supervisor Lars Hammarstrand for his guidance and insightful input during the project. We would also like to thank all of the members in the DAT and HADES teams at Zenuity for all their help, support and encouragement. A special thanks to our supervisor at Zenuity, Henrik Arnelid, and to out team-mate Erik Karlsson. Their knowledge and data processing expertise has been of great help. Last but not least we would like to thank Christian Klinteberg, Ellen Korsberg and Eliza Nordén for all the lunch walks, coffee breaks and pep talks at Zenuity.

Lovisa Hagström, Lisa Sjöblom, Gothenburg, June 2019

Contents

List of Figures

1
Introduction

Autonomous vehicles are on the verge of becoming a part of our day to day life. The vehicles on the market today have an increasing number of Advanced Driver Assistant Systems (ADAS) features, such as adaptive cruise control, lane keep assist and park assist [1, 2]. In the meantime, several companies such as Tesla, Uber, Google and Zenuity are actively working to solve the problem that is autonomous driving (AD). Undeniably, ADAS and AD systems do already, or will in the future, facilitate transportation for a magnitude of people.

While the development of ADAS features — along with completely self driving cars — of course is a challenge in itself, there is still a long way left before society gets on board to support the increasing technical development. Many companies find themselves at a stage where the need for necessary permissions and legislation to test their products is a huge bottleneck. There is also a need to build public *trust* in autonomous vehicles [3]. Not to mention, companies in the ADAS and AD fields need to be certain that their products are *safe*. For these reasons, the industry needs to develop trustworthy and safe tests for their products. On those grounds, verification and validation of ADAS and AD systems are imperative for the development of autonomous vehicles.

As empirical verification is costly, sometimes not permitted, and, most importantly, unfeasible [4], *virtual verification* is necessary. To properly verify that a driver system is safe, all or a majority of all possible driving scenarios need to be checked. While it is impossible to empirically drive through all possible scenarios, virtual systems can simulate through them in a quicker manner. Consequently, reliable virtual verification methods are crucial for a safe development of ADAS and AD systems.

To enable simulated tests and virtual verification of an AD system, accurate models of the sensors that are used by the system are needed. As the sensors are the *eyes* of an AD system, the information they would output for a driving scenario needs to be simulated when the scenario is simulated. To this end, a sensor model that takes a certain driving scenario as input and gives the corresponding sensor information for the scenario as output, would be very useful.

Recent research has indicated that artificial neural networks may be used to model sensors used for autonomous driving [5]. Since neural networks are excellent function

approximators, they can learn to accurately map different inputs to different desired outputs. Within the field of *deep learning*, neural networks have been used for a wide range of tasks, from image enhancement, art generation, article writing and emotion detection in text, to writing cooking recipes. Hence, it is hardly a surprise that neural nets also have proven themselves useful for sensor modelling.

In this thesis, models of a radar sensor are developed based on state-of-the-art deep learning algorithms. The purpose of each model is to facilitate virtual verification of ADAS and AD systems. To develop the models, the neural networks *Variational Autoencoders* and *Generative Adversarial Networks* are utilised. Also, to enable training of the networks, a dataset is created. The thesis is performed at Zenuity, using their data and data processing frameworks.

1.1 Thesis Objective

The aim of this master thesis is to create a stochastic model of the detection level output from the radar sensors used by Zenuity. To do this, deep generative models are used. The generative model should be able to produce realistic radar readings for a given situation. This amounts to creating output that is as similar as possible to what the real radar sensor would, both in terms of accuracy and flaws. This model does not consider time series of the data, but instead views and predicts the output for one instance at a time.

The input to the model is an environmental description. This data, called the conditional input, is given by object detections from a lidar. When using the trained model, it should also be possible to input synthetic object level lidar data. The networks are trained on processed radar data at detection level, corresponding to the environmental descriptions given as input.

In essence, the goal of this thesis is to find the function f, or an adequate approximation of f, that gives

$$y = f(c),$$

where y is the detection level output from the radar sensor and c is the conditional input, containing information about the environment and objects surrounding the radar sensor. Thus, c can be provided by some other sensor that can interpret the environment, or it can be synthetically created in some simulation software. For this thesis, a lidar provides c.

1.2 Related Work

Several approaches to radar modelling exist. One approach is to directly calculate the electromagnetic wave propagation and reflection of objects [6]. Another more black-box approach for radar modelling is to use deep generative neural networks, such as Variational Autoencoders and Generative Adversarial Networks, that are trained to imitate the output of a real radar [5, 7]. Presently, the standard method for radar modelling utilises statistical models [8], as the previously mentioned methods are fairly new and still in development. For all radar modelling purposes, no matter the approach, computational time and model accuracy are of importance. Since a high model accuracy generally entails a long computational time, a common problem with radar modelling is the trade-off between high accuracy and short computational time.

One example of radar modelling based on the standard radar modelling approach, that is, the statistical approach, is given by Bühren and Yang [8]. They created a radar model based on a statistical analysis of radar data, such that e.g. measurement errors in distance and relative speed were simulated with white Gaussian noise, according to their observations of measured data. For the evaluation of their results, Bühren and Yang visually compared readings from the model and the real radar. They found that simulated and real data displayed similar characteristics, even though they did not consider every physical effect for the radar modelling.

As previously mentioned, neural networks have already been used for the modelling of an automotive radar sensor. The main advantage of using neural networks for radar modelling is that they have the potential to require less computational power, while still maintaining the high accuracy otherwise associated with ray tracing and long computational times [6]. Since artificial neural networks are capable of generalisation, they have the capacity to model a radar given sufficient training data.

Neural networks, such as Variational Autoencoders and Generative Adversarial Networks, along with two baseline models, were implemented by Wheeler et al. to model a radar sensor [5]. The input to the models was an environmental description and the modelled output was a radar power field prediction. The environmental description was formatted as an occupancy grid, i.e. a grid that indicates where surrounding vehicles, buildings, barriers etc. are located. Wheeler et al. found that the models based on the Variational Autoencoders and on the Generative Adversarial Networks had a better performance than the baseline models, while a model that combined the two former mentioned networks had the best performance.

Conditional Variational Autoencoders were used by Suhre and Malik for modelling a radar sensor [7]. However, they modelled the output at object level. The output gave the properties of the objects such as 2D position, velocity and heading of detected objects. The input was of the same format as that of Wheeler et al. For a model with 10 layers, Suhre and Malik obtained reasonable results for the errors

of measured object properties, for different automotive scenarios that contained at least one target vehicle.

Generative adversarial networks have a wide range of application areas beyond radar modelling. While the desired input and output of each adversarial net may vary, knowledge about the network from one area may be valuable across applications. Such knowledge was presented by Sajjadi et al. when they created a deep convolutional generative adversarial net *EnhanceNet*, for image enhancement [9]. The input to the net was a 32×32 photo and the output a 128×128 photo with a higher resolution. As such, the architecture of the EnhanceNet was largely based on convolutional layers and residual blocks. The resulting net proposed by Sajjadi et al. proved to be very capable of deriving high resolution features for various images.

1.3 Contributions and Thesis Outline

Altogether, the contributions of this thesis are the creation and evaluation of a dataset and a radar model. The contributions mainly consist of three parts. The first part relates to the formatting and analysis of a dataset for the radar model development, while the second part addresses the model development using neural networks. The third part outlines the evaluation of obtained model results.

Firstly, a dataset is created for the radar model development. Data from both radar and lidar is collected and formatted to suit an occupancy grid format. The method and a description of the format of the generated datasets is given in Chapter 3. Furthermore, an analysis of both overall properties and instance properties of the gathered datasets is provided in the chapter. The data format is similar to that of Wheeler et al. [5], such that polar occupancy grids are formed to give environment descriptions and radar model output. Differently from Wheeler et al. a more detailed data environment description with several grid layers is used in this thesis, without object lists.

Secondly, different model architectures are developed to model a radar sensor. Both Conditional Variational Autoencoders (CVAE) and Conditional Generative Adversarial Networks (CGAN) are implemented together with different training algorithms. The CVAE architecture is largely based on the radar modelling work of Wheeler et al. and the CGAN architecture is mostly based on that of the EnhanceNet by Sajjadi et al. in the sense that it uses residual blocks to map an input to an output. The underlying theory that motivates the model architectures and algorithms used in the project is provided in Chapter 2, while the architecture descriptions of the models, along with their training algorithms, are found in Chapter 3.

Thirdly, the thesis results are evaluated with the evaluation methods described in Chapter 3. Both derived radar models and dataset properties are considered here. The corresponding results for the radar models are given in Chapter 4. The overall

performance as well as the behaviour for a few different scenarios are evaluated for the radar models, and the radar model performance is compared to that of the real radar sensor. Moreover, the results are discussed in Chapter 5, where the different model architectures are compared, improvements are proposed and the model performance is set into perspective with the quality of the data. The thesis ends with a summary of the conclusions of the outcome of the project, given in Chapter 6.

2

Theory

The generative models developed in this master's thesis are based on artificial neural networks, described in section 2.1. The implemented model types, the Variational autoencoder and the Generative Adversarial Network, are explained in section 2.2 and section 2.3. Moreover, lidar and radar sensors, used to obtain training data for the models are described in section 2.4.

2.1 Artificial Neural Networks

Since ANNs are powerful function approximators, they have been utilised in a multitude of fields; from image recognition, bug hunting in code, to music generation [10, 11, 12]. Typically, ANNs consist of several connected *artificial neurons* which can be trained with the backpropagation algorithm [13]. During training, the parameters of the neurons are tuned to suit a given input and desired output of the ANN. The neurons can be arranged into layers, such as fully connected layers or convolutional layers, described in section 2.1.2. The layers, and their respective activation functions, make up the network. Several networks and models derived from the ANN exist, such as *Deep Neural Networks* and *Deep Generative Models*, which will be further described in sections 2.1.1 and 2.1.4.

2.1.1 Deep Neural Networks

Deep Neural Networks (DNNs) are artificial neural networks that can learn data representations at higher levels, through *deep learning* [14]. This deep learning is made possible by the use of several hidden layers within the ANNs, as well as large amounts of data. Some of the initial state of the art DNNs used for image recognition, such as the VGG nets and GoogLeNet, have 16-19 and 22 hidden layers respectively [15, 16]. The higher levels of data representation that are achievable with deep learning have been found to be very useful not only for image recognition, but also for e.g. speech recognition, the analysis of particle accelerator data and

natural language understanding [14].

However, while DNNs are powerful in the sense that they can learn very complex functions, they also suffer from more issues during training than shallow NNs. More layers lead to more weights in the network in need of tuning, and the backpropagation algorithm needs to go deeper into the network, resulting in a more difficult training process. Apart from taking a longer time to train, there are also issues with exploding or vanishing gradients in the DNNs [17] and internal covariate shift, described by Ioffe and Szegedy [18].

Consequently, considerable efforts have been made to find methods that alleviate DNN training. For example, adding batch normalization between each layer in a DNN has been found to prevent an internal covariance shift, such that it both speeds up the training, and leads to a better network performance [18]. Furthermore, a different weight initialisation for the DNN can improve its training [17]. A final example of how to improve upon DNN training and performance is to make use of residual nets, such that the layers instead learn residual functions with reference to the layer inputs, as seen in Figure 2.1.

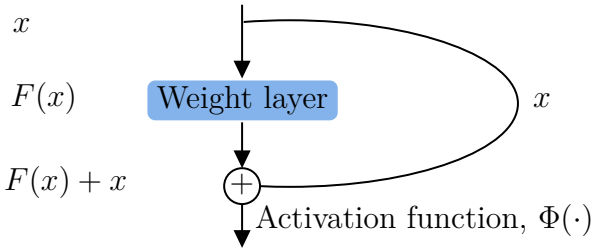


Figure 2.1: Illustration of a residual block. The weight layer is given an input x for which it is trained to map the residual $F(x)$. The final output from the model is then given as $\Phi(F(x) + x)$.

The hypothesis is that it is easier to optimise the residual mapping than to optimise the original, unreferenced mapping. For example, it becomes easier for the weight layer to perform an identity mapping in this setting. With this approach, DNNs with up to 152 layers are possible to train. One such net, created by He et al. won the ImageNet Large-Scale Visual Recognition Challenge in 2015 [10].

2.1.2 Convolutional Layers

Convolutional layers in a DNN are of great use for the analysis of multi-dimensional data samples for which the ordering of the data points matters. Examples of such samples, consisting of ordered data points, are images and radar detection grids. In the case of an image, the data points are given by pixels, and in the case of a radar detection grid, the data points are given by ones (for a detection) and zeroes (for no

detection). If the ordering of these points was to be altered, the original image or radar detection grid would be lost. Convolutional layers have found a widespread use within for example image recognition and language processing [10, 16, 19].

The principle of convolutional layers is illustrated in Figure 2.2. The main idea behind the layers is to sweep filters, that are of a smaller dimension than the input, over the input data such that each output point is calculated from a subset of the input data. Each subset consists of points adjacent to each other. In Figure 2.2, each subset consists of nine points from a filter of dimension 3×3. This way, the ordering of the data points in the input is taken into account [20].

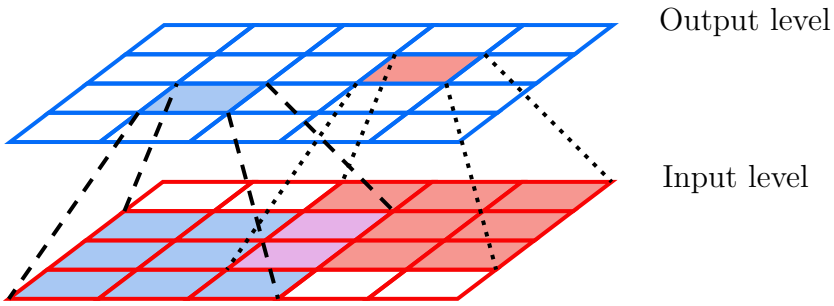


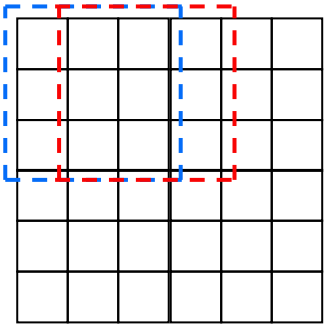
Figure 2.2: An illustration of the general idea behind a convolutional layer. The coloured squares on input level correspond to the identically coloured squares on output level. As such, nine squares on input level may correspond to one square on output level for a convolutional layer.

Convolutional layers can both be used such that they preserve the dimension of the data, as seen in Figure 2.2, or such that they reduce the dimension of the data across the layer. The latter case can for example be achieved by taking fewer subsamples from the previous layer, resulting in downsampled data. These fewer subsamples are taken by specifying the number of *strides* that the filter should take between each subsample, as illustrated in Figure 2.3. The stride determine over how many subsamples the filter is applied to. If the dimension is to be preserved for a 2D data grid, a stride of $(1, 1)$ is specified, and if the dimension is to be decreased, a stride of for example $(3, 3)$ can be specified.

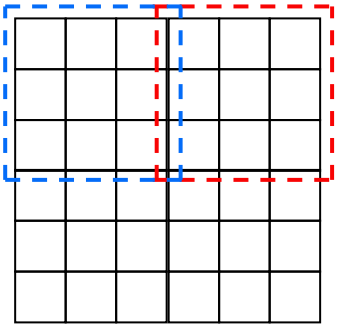
Furthermore, each element in a filter can be weighted, such that a filter can be adapted to search for a certain feature within each covered subset, as illustrated in Figure 2.4. Then, when the filter is swept over the input data, an output is generated that for each subset specifies how well represented the feature that the filter searches for is in each subset [20]. Consequently, convolutional layers can be used to search for shapes and patterns in visual data, for example.

Convolutional layers can be stacked in networks that then are called *Convolutional Networks*. A convolutional layer in a neural network usually utilises several filters in the same layer. Then, all filters in the layer are applied to each subset of the data. For a network consisting of convolutional layers, the weights in the filters of

the layers make out the trainable parameters.



(a) Stride (1,1).



(a) Stride (3,3).

Figure 2.3: An illustration of how strides can be specified for convolutional filters. A filter is swept over the input with different strides between the subsamples, resulting in outputs of different sizes. For (a) with stride (1,1) the output dimension will be the same as for the input, (9,9). For (b) with stride (3,3), the output dimension will be (2,2).

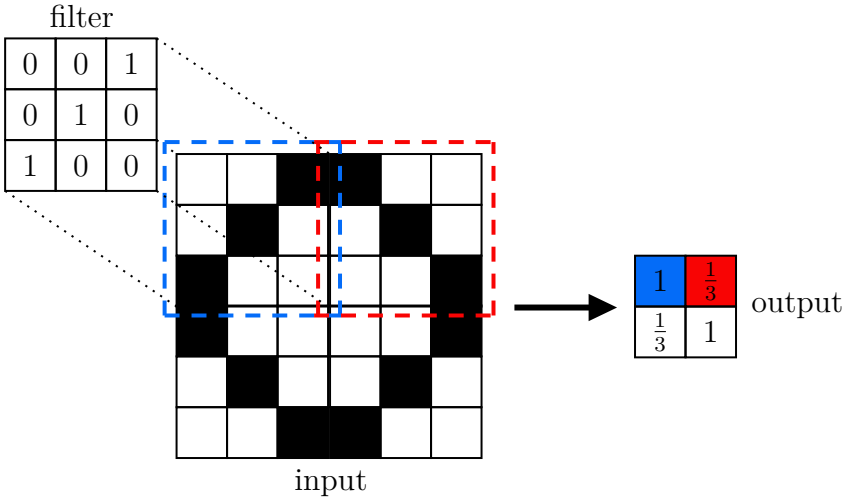


Figure 2.4: An illustration of how a filter works for a convolutional layer. A filter is swept with a stride of (3,3) over an input. The input consists of white and black pixels, where white corresponds to a value of zero, and black corresponds to a value of one. The convolutional layer generates an output that indicates which parts of the input that contain the feature that the filter has learnt to map.

2.1.3 Transposed Convolutional Layers

Transposed convolutional layers, also called *deconvolutional layers*, are similar to convolutional layers, while they can be used to up-sample data [21]. The principle

behind deconvolutional layers is illustrated in Figure 2.5. Here, the squares on the input level correspond to the squares on output level with the same colour. Hence, a single value in the input layer is given as input to several values in the output layers.

Similarly to Figure 2.4 for the convolutional layer, the input and output of the deconvolutional layer are described in Figure 2.6 for one filter. Here, one input point is used to generate several output points, guided by the filter. Consequently, a more complex output is generated from a simple input. In this way, data is up-sampled to suit certain features.

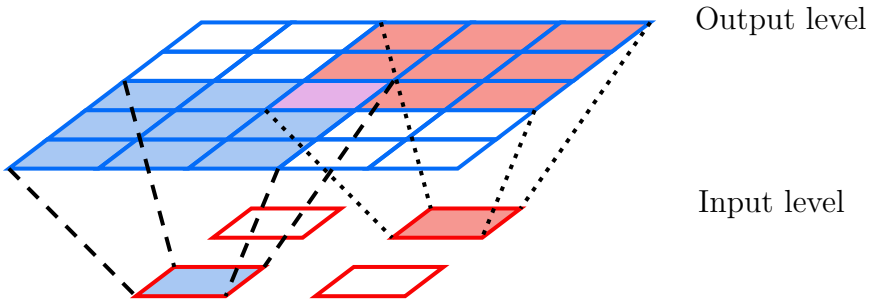


Figure 2.5: An illustration of the general idea behind a transposed convolutional layer. The coloured squares on input level correspond to the identically coloured squares on output level. As such, one square on input level may correspond to nine squares on output level for a transposed convolutional layer.

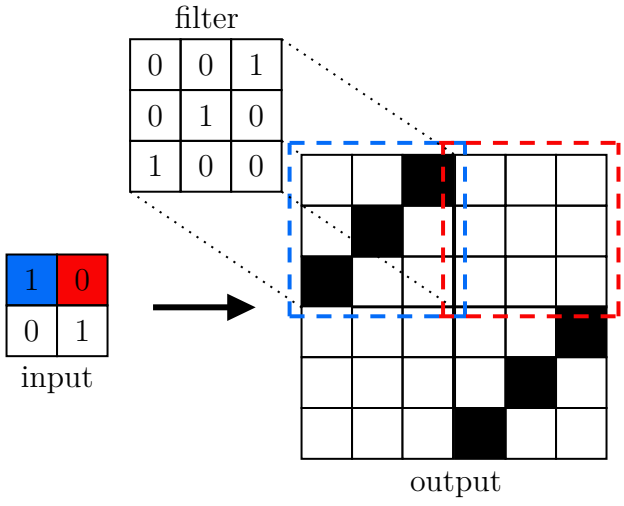


Figure 2.6: An illustration of how a filter works for a transposed convolutional layer. A filter is swept with stride (3,3) to create an output. The output consists of white and black pixels, where white corresponds to a value of zero, and black corresponds to a value of one. The convolutional layer generates an output that contains the feature that the filter has learnt to map based on an input that describes where the feature should exist.

The transposed convolutional layers can be used to artificially up-sample from a low resolution to a high resolution, as is done in [9]. This type of layer can also be used to generate images from one dimensional data, such that noise can be used to generate images from a Generative Adversarial Network [22]. This will be explained further in section 2.3.

2.1.4 Deep Generative Models

Generative modelling is a branch of machine learning where a model is trained to generate new data. The new data should be similar, but not identical, to the data that was used to train the model. To do this, the model aims to learn the distribution of the data, after which it can produce new data points from this distribution. Some models learn an explicit form of the distribution, in the form of hyperparameters, from which new samples can be drawn. Others generate implicit representations of the data distribution, such as the ability to directly generate samples from it [23]. Deep neural networks can be used to create generative models, so called deep generative networks [24].

There are several types of generative models based on deep learning, such as deep Boltzmann machines and deep belief networks [23]. This category also includes the two popular approaches generative adversarial networks (GANs) and variational autoencoders (VAEs) [22, 25]. As previously mentioned, the two latter networks will be used in this project to model the radar sensor. Therefore, GANs and VAEs will be further described in the sections below.

2.2 Autoencoder Networks

An autoencoder is trained to attempt to copy its input, y, to its output, y'. It does this by representing the input with a code, described by an internal hidden layer, $z = f(y)$. The output is then obtained from the mapping $y' = g(z)$. The part of the network that handles the input representation aspect is called the encoder and the part that generates the output by mapping z to y' is called the decoder. Figure 2.7 illustrates the parts of an autoencoder.

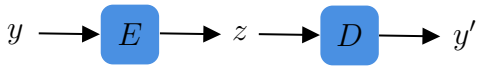


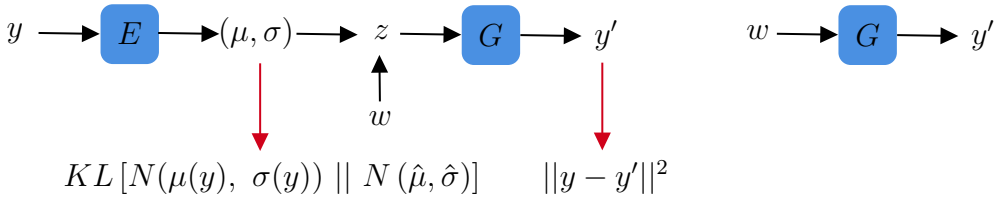
Figure 2.7: The basic schematics of an autoencoder. E denotes the encoding layers and D the decoding layers. y is the input to the model, and the goal of the network is to copy produce an output y' that is as similar as possible to y. z is the values of an internal hidden layer.

Ideally, the goal for the encoder is to map $y = y'$. However, a network that performs a mere copying task by setting $g(f(y)) = y$ everywhere is of little use. Instead some form of restriction is applied to the network, for example to the dimensionality of z, to prevent it from generating perfect copies. The aim is that the applied restrictions will force the model to prioritise which aspects of the input to replicate, such that useful properties of the data are learned [23]. In this way, the autoencoder can for example be used for feature learning or dimensionality reduction.

2.2.1 Variational Autoencoders

The variational autoencoder has many similarities with a basic autoencoder, but with the important difference of being a generative model. Hence, the decoder will from now on be called generator, denoted G. The goal of the VAE is to be able to generate new samples, that are similar, but not identical, to the training data. Here, the internal layer z is called the latent variable. In essence, after training the VAE, the generator can be extracted and used to generate new samples by feeding it z.

When using the trained VAE to produce new data, a way of knowing what z to feed to the generator is needed [25]. In other words, the distribution of reasonable z needs to be known. This is solved by applying some restrictions to the latent variables in the network, and thus a neural network capable of generating data similar to the training data can be created. The model therefore asserts that z can be drawn from a simple distribution, for example $N(\hat{\mu}, \hat{\sigma})$ [26].



(a) Variational autoencoder. (b) Generative network.

Figure 2.8: The basic schematics of a variational autoencoder can be seen in (a). E denotes the encoding layers and G the generative layers. The two loss terms that the VAE is trained on are indicated by red arrows. w is sampled from a normal distribution $N(\hat{\mu}, \hat{\sigma})$. z is computed as $z = \mu + \sigma w$. In (b), the parts of the VAE used for generating new samples are depicted.

The key insight of this idea is that any distribution in d dimensions can be generated by taking a set of d variables that follow a normal distribution, and mapping them with a sufficiently complicated function. Since neural networks are powerful function approximators, they can form this *sufficiently complicated function*. Thus, all that is needed is to add the generation of standard normal distributed values w to the

network, and some extra layers for the mapping of these w to our latent variables z, as in Figure 2.8a. During training, the encoder is then taught how to generate z from w, with the help of the training data. As a result, the network is also taught to find a function $Q(z|y)$ which can take a value of y and give a distribution over z values that are likely to produce this y. Usually, it is assumed that $Q(w|y) = N(w|\mu(y), \sigma(y))$, where μ and σ are parameters that are learned from the data by the network.

Consequently, the training of the VAE involves the goal of maximising the likelihood for the data y under the generative model, i.e. to maximise $P(y)$, as well as to make Q produce representations for y that can be reliably decoded by the generative layers. With some assumptions about Q and further derivations based on probability theory, the full equation that is to be optimised for the VAE boils down to

$$E_{y\sim S}\Big[E_{w\sim N(0,I)}\Big[\log P(y|z = \mu(y) + \Sigma^{1/2}(y) \cdot w)\Big] - \mathcal{D}\Big[Q(z|y)||P(z)\Big]\Big], \qquad (2.1)$$

where the data y is sampled from a dataset S and $\mathcal{D}$ denotes the Kullback-Leibler divergence (KL-divergence). This divergence denotes the difference between z and the distribution that z is restricted to. The goal is to maximise the expression in Equation (2.1), and this is done by maximising $P(y)$ as well as minimising the KL-divergence. Thus, during training of a VAE, two errors are to be minimised; the original error for the autoencoder for the mapping of input to output, and the error derived from the KL-divergence [26]. The last error may be intuitively interpreted as a means to train the part of the network that has been added for the mapping of normal variables, w, to latent variables z.

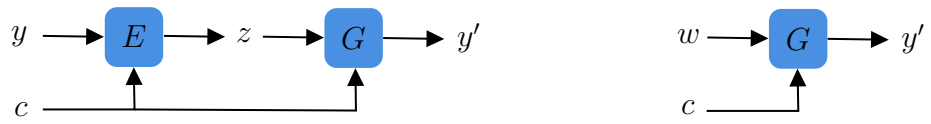
After training, the generator is extracted and used to generate new data samples. See Figure 2.8b for a specification of the parts of the VAE that are kept for the data generation. w is sampled from a normal distribution and fed through the generator, resulting in the output y'.

2.2.2 Conditional Variational Autoencoders

A conditional variational autoencoder (CVAE) is a modified version of a VAE. Using variational autoencoders as described above, new data samples can be generated from a normal distribution. However, some additional properties of the output are often desired. Consider an application in computer graphics, where a part of an image is missing, and the missing pixels need to be approximated. Since there are many possible solutions for this problem, a simple regression model will not do. This would result in a blurry image, since the model would try to average between all possible solutions. Instead, a model that learns a distribution over possible solutions to sample from is needed, and a variational autoencoder comes to mind. For this problem, a CVAE can be used.

CVAEs condition the generative process on some input. In the example above, the conditional input could for example be the image that is missing data. The input

y would then be the missing pixels, that the network should find. In practice, this additional restriction is achieved by inputting the conditional input to both the encoder and generator. The setup is described in Figure 2.9. The conditional input enables far more sophisticated areas of use, such as image generation of desired objects and features [27].



(a) Conditional variational autoencoder. (b) Generating part.

Figure 2.9: The structure of a Conditional Variational Autoencoder can be seen in (a). E denotes the encoding layers and G the generating decoding layers. The conditional input c is fed to both the encoder and the decoder. The loss functions and sampling are identical to the ones in the VAE, but are left out here for readability. As can be seen in (b), only the decoding layers (the generator, G) are used to generate new samples.

The conditional input is different from the normal input y in the sense that it is not trained on, i.e. it is not a part of the loss function. It is kept as input to the generator when the training is completed. The purpose of the conditional input is to give the network a hint of what the output y' should look like when generating new samples. Hence, when using conditional input to VAEs, the conditional input data must correspond to the regular input data in some way.

2.3 Generative Adversarial Networks

Generative Adversarial Networks were first introduced by Goodfellow et al. in 2014 [22]. The basic idea behind GANs is to train two networks in parallel, the first being a generative network, G, and the second a discriminative network, D, see Figure 2.10. Since their introduction, GANs have been used for numerous purposes that require data generation.

Instead of learning from a cost function, the networks G and D in a GAN learn from each other. This lack of a heuristic loss function enables the generative network to learn a more general distribution. For example, in the case of image generation, a heuristic loss function would be a pixel-wise mean squared error, which does not necessarily perform well in evaluating the credibility of images. Instead, a discriminator can be taught how to distinguish between real and fabricated images. This capability of learning data representations have made GANs useful within for example art generation, artificial face generation and artificial face ageing [28, 29, 30].

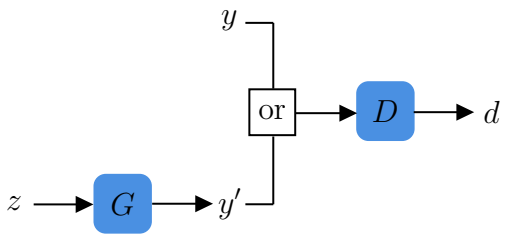


Figure 2.10: A basic illustration of a GAN. G is a neural network that generates samples y' from an input noise z. D is a neural net that takes in data samples, either true, y, or generated, y', for which it assigns a probability, d, of the sample being true.

More formally, G generates samples $y' = G(z)$ based on input noise variable z. D outputs the probability $d = D(y')$ of a sample y' being auhentic, i.e. not generated by G. D is then trained to correctly label both true and generated samples, while G is trained to minimise $\log(1 - D(G(z))$, i.e. to fool D that generated samples are authentic. Consequently, D and G play the following two-player minimax game with value function $V(G, D)$

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z))]. \tag{2.2}$$

However, this original formulation may not be optimal in the regard of training G to minimise $\log(1 - D(G(z))$. Because D usually is very certain about which samples are generated in the initial phase of training, we will have that $D(G(Z)) \approx 0$ for this phase, and since the derivative of $\log(x)$ becomes smaller as x is increased from zero to one, this results in a mostly unchanging $\log(1 - D(G(z))$ value for the initial training phase. Hence, G can instead be trained to maximise $\log D(G(z))$, since it provides a better initial gradient for G while it still leads to the result defined by Equation (2.2) [22].

Given that the two networks in a GAN have enough capacity for their designated tasks, D and G can reach their respective optima. E.g. the networks have enough capacity if they each have an accurate model architecture capable of solving their respective tasks. At the optima of G and D, G will generate samples that are indistinguishable from the real data, and D will be optimal at differentiating false from true samples. However, when G is at its optimum, $G^*(z)$, even an optimal D should be unable to correctly identify generated samples. E.g. we should have that the optimal D^* is completely uncertain as to whether a given generated sample from the optimal generator, $(y')^* = G^*(z)$, is false or true, such that it predicts a 0.5 probability of the sample being either true or generated, $D^*((y')^*) = 0.5$ [22].

To summarise a GAN, the purpose of G is to train on and learn from a dataset in order to produce new similar data samples. The purpose of D on the other hand is to evaluate a given data sample and deduce whether it comes from the dataset or

the generator. Then, G is trained to maximise the probability of D believing that generated samples come from the dataset and D is trained to learn how to tell the samples apart. The structure can be likened to a situation where G works as an art counterfeiter and D as an art specialist, where G tries to fool D with its faked artwork and D tries to spot the fakes. The idea is then that the two networks train each other until the fake samples and the real ones are indistinguishable from one another. Once the networks have converged, the discriminator can be discarded and the generator is taken as the final generative model.

2.3.1 Convergence of Generative Adversarial Networks During Training

In the previous section, it was stated that a GAN has an optimum. However, quick convergence to this optimum is not guaranteed, as the networks may suffer from convergence issues during training. The stabilisation of the learning process in GANs is still an open problem, and they are generally difficult to train. There is little to no theory explaining the unstable behaviour of GAN training, and many recent papers on GANs are dedicated to heuristically finding stable architectures for GANs [31].

The main issue during training of a GAN is that the discriminator may *saturate*, such that it becomes very certain of which samples are fake and which are true. I.e. instead of being approximately certain, the discriminator is almost binary in its probability output, only generating values close to either zero or one. Initially, one may think that a very certain discriminator gives good feedback to the generator. However, this is not the case. Instead, the updates from the discriminator to the generator become increasingly worse as the discriminator gets better [31]. Since the generator is trained only on feedback from the discriminator, saturation of the discriminator implies that the training of the GAN fails. Some of the heuristically found methods to avoid saturation and to stabilise the general training of a GAN will be discussed in the section below.

2.3.2 Stabilisation Methods for the Training of Deep Convolutional Generative Adversarial Networks

Deep convolutional generative adversarial networks (DCGANs) are GANs that utilise hidden convolutional layers in the generator and the discriminator. These types of networks have found several uses within 2D-data generation and are used for e.g. resolution enhancement of images [9] and radar modelling [5]. However, as they are deep networks, their training process is not straightforward and convergence is not guaranteed.

Given the potential of these types of networks, significant efforts have been made to discover methods that stabilise their training. One article in particular describes several steps that have been found to both speed up and stabilise the training of DCGANs [32]. Some of these steps are to

- use leaky ReLu as activation function in the discriminator to improve the training gradient for the generator
- add batch normalisation between each layer in the networks, with the exception of the output of G and the input of D
- avoid fully connected hidden layers within the networks, to speed up their training.

Other methods to improve the training process of DCGANs are to use residual blocks, as in the case for DNNs, and possibly to make the whole network model a residual. This was done for resolution enhancement with a DCGAN called *EnchanceNet* [9], where a residual was mapped and then added to an image that already had been enhanced with existing standard algorithms, as illustrated in Figure 2.11. The residual model in the figure is similar to the residual block in Figure 2.1, with the difference that it maps the residual between the model input and output, instead of only over one or a few layers.

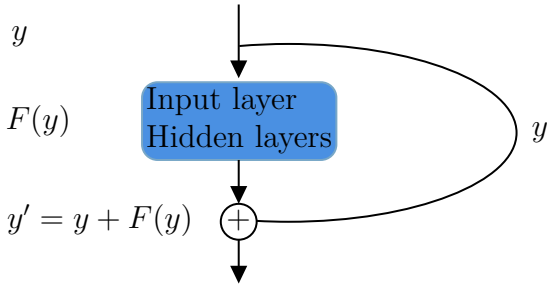


Figure 2.11: A model that is trained to map the residual $F(y)$ to an input y in order to generate a final output y'. Thus, the model only has to learn the residual $F(y)$. A structure similar to this was employed in the EnhanceNet [9].

Another method employed by the EnhanceNet to improve the training process of a DCGAN was to adapt the training steps for D and G based on the current accuracy of the discriminator during training. Consequently, the discriminator was only trained if its accuracy fell below certain thresholds for real respectively fake data samples. In that way, saturation of D as a consequence of too much training could be avoided.

2.3.3 Conditional Generative Adversarial Networks

An additional conditional input, c, can be given to both the generator and the discriminator in a GAN, resulting in a conditional generative adversarial network (CGAN) that can direct the generated data towards desired features [33], see Figure 2.12. The CGAN differs from a basic GAN such that the samples generated from it are not only based on some arbitrary noise input. For example, if a generator is trained to generate pictures of bedrooms, the conditional input can be a description of desired bedroom features, such as *blue* or *large*, on which G can be trained to generate images from.

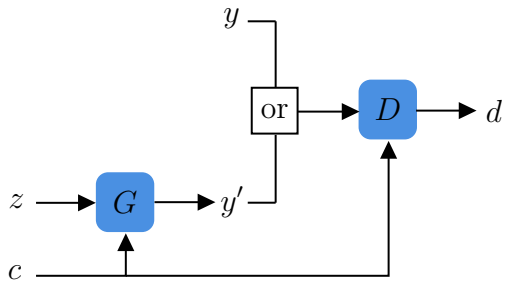


Figure 2.12: Illustration of a conditional GAN. Both G and D receive an additional conditional input c. G is a neural network that generates samples y' from an input noise z and the conditional input. D is a neural net that takes in c as well as data samples, either true, y, or generated, y', for which it assigns a probability, d, of the sample being true based on the conditional input.

Thus, the two-player minimax game for a conditional GAN has the objective function,

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x|c)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z|c))],$$

where c denotes the conditional information. This function is the same as that of the simple GAN in equation (2.2), with the only difference that the distributions in the function are conditioned on c.

2.4 Data Collecting Sensors

The sensors used for this thesis are radars and lidars. The sensors share the purpose of perceiving their surrounding environment and generating data that describes it. In most other regards, the sensors are very different. For instance, they differ in range of perception, basic workings, resolution and dependence on weather conditions. In order to understand the data generated from the sensors and to know what to expect from it, essential sensor theory will be described in this section.

2.4.1 Radar Sensor

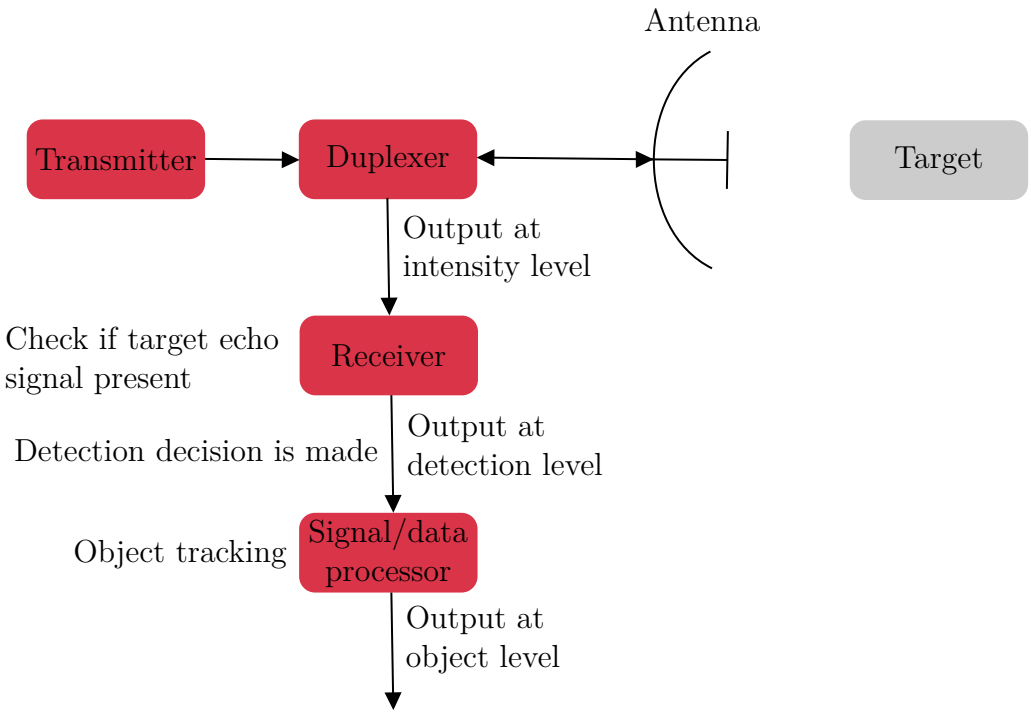


Figure 2.13: A schematic view of the basic processing steps for a radar. Note that the data processing and object tracking are performed over time while the receiver operates for separate time instances.

A radar sensor is an electromagnetic sensor used for detecting, locating, tracking and recognising objects of various kinds. The sensor can operate at both short and long target distances, where the targets for example can be aircrafts, ships, spacecrafts, automotive vehicles, insects and rain. Radar sensors have the potential to determine presence, location, velocity and shape of the targets. The radar is distinguished from other optical and infrared sensing devices by its ability to detect faraway objects also under harsh weather conditions, such as heavy rainfall or snowfall, and by its ability to determine their distances with precision. The radar was first developed during the 1930s and 1940s, mostly in correlation with the second world war [34].

A radar sensor operates by radiating a beam of electromagnetic energy from an antenna. The beam scans a region where targets are expected. When a target is illuminated by the beam, it intercepts some of the radiated energy and reflects a portion back toward the radar system. This results in radar readings at intensity level. After this, a decision is made by a receiver at the output of the radar antenna as to whether or not a target echo signal is present. Present targets result in readings at detection level by the radar, occupying sensor resolution cells, where the readings for example can cover range. Readings at detection level, so called reflection points, can then be tracked over time with object tracking algorithms. This results in a list of objects and their specific features over time, such as velocity and position [35].

These tracked objects can for example be automotive vehicles. See Figure 2.13 for a schematic view of the processing steps for a radar.

As mentioned previously, radar readings at detection level are tracked over time with object tracking algorithms to obtain an output at object level for the radar sensor. For object tracking based on sensor measurements, *Multiple Target Tracking* (MTT) is used, i.e. the process of successively determining the number and states of multiple dynamic objects based on noisy sensor measurements [36]. The object level radar data can then be used for example for simulation tests of the decision making of ADAS and AD systems.

For object tracking on radar sensor data within automotive applications, there are some basic limitations and issues. These relate to e.g. range, coverage, and typical sensor errors. In general, a radar sensor can also display very complex and counter intuitive behaviours. Some of these issues will be described in more detail in the following sections.

Firstly, the radar sensor has a limited coverage depending on its frequency and type; a lower frequency generally results in a larger range, while a higher frequency yields a smaller range. For example, radar sensors with a frequency of 77 GHz are used for obstacle detection in autonomous driving [35]. These sensors can also be of different types, which may prioritise either a long range or a wide view.

Furthermore, the readings from a radar can display different errors. For example, objects can be completely missed by the radar system, or false detections, that is false positives, can occur. A false detection, also called *false alarm* or *detection of ghost target*, occurs when a radar system registers a reading for an object that does not exist, or for an insignificant object that should be ignored [37, 38]. For the automotive radar, this may be caused by multi-bounce-reflections on metal objects such as barriers near the road. These types of errors may then propagate into object level, resulting in missed objects or ghost objects [39].

Another example of complex radar behaviour is that a radar sensor also can be limited by occlusion, such that it may not be able to detect an object if it is hidden behind something, for example a car. Occlusion and ghost targets make out the more unexpected and undesired behaviour a radar can display [39].

Lastly, the detection performance also depends on the properties of the object to be detected. For example, the size, shape and material of an object all affect the probability of detection for a radar [40]. For example, an automotive radar sensor is much better at detecting road barriers made of steel than barriers made of concrete. As such, the radar sensor may fail to detect concrete barriers, while it rarely misses steel barriers.

2.4.2 Lidar

Lidar, or *light detection and ranging*, is a sensor commonly used in the development of autonomous vehicles. As depicted in Figure 2.14, the sensor uses laser beams to determine the distances to surrounding objects by measuring the time it takes for each beam to be reflected. This results in observed reflection points that form a point cloud of the surrounding environment, as seen in Figure 2.15. Based on this point cloud, detections of e.g. vehicles, pedestrians and barriers can be made at each time instance. These detections are then used by a tracking algorithm, which connects them between time instances and approximates true positions, velocities and other properties.

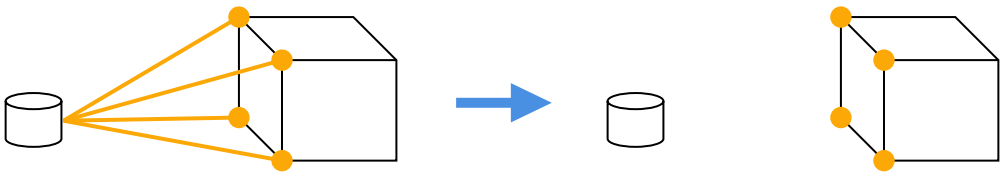


Figure 2.14: The lidar, that is the small cylinder, sends out laser beams, marked in yellow, which are reflected by the cube object. This results in a point cloud.

A lidar works well in all lighting conditions, and it is very accurate in comparison to automotive radar sensor. However, the performance of the lidar is very sensitive to environmental influence, such as rain or snow [38]. It also typically has a shorter range than for example the radar sensor. Generally, the lidar is considered to be too expensive to include in the production of self-driving vehicles.

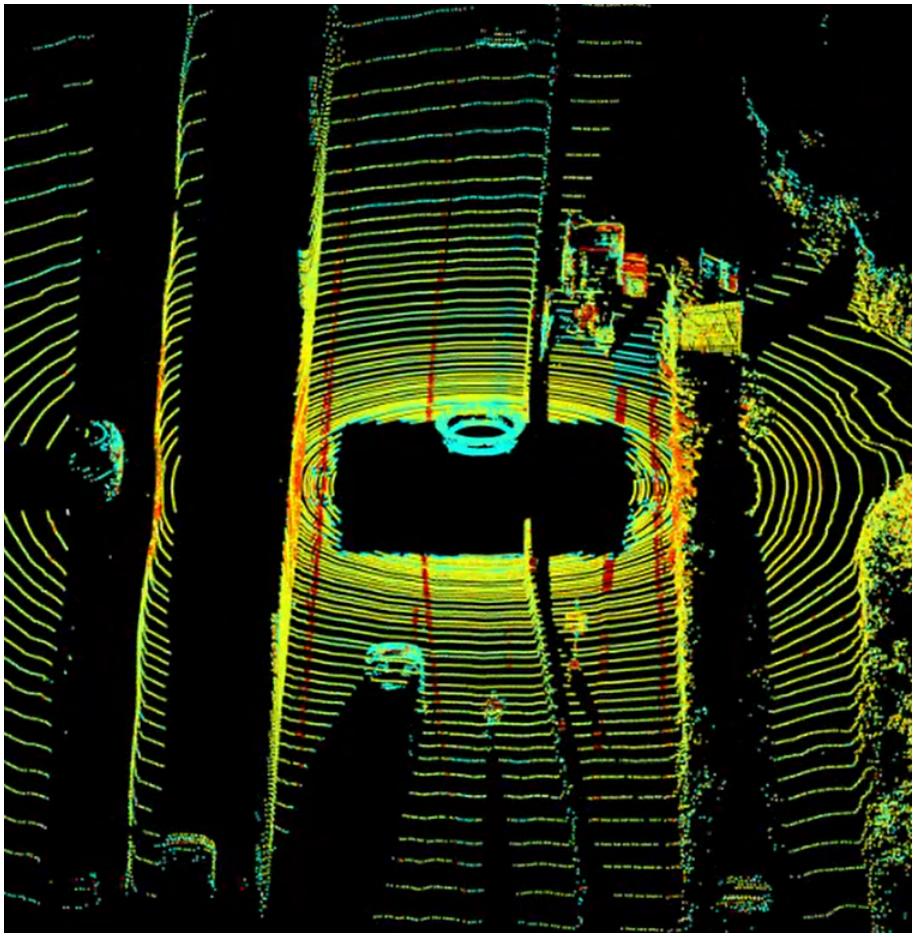


Figure 2.15: An example of a lidar point cloud. An ego vehicle with a lidar on the rooftop is seen in the centre of the figure. Surrounding the ego vehicle is a very dense point cloud that describes the shapes of e.g. cars, buildings and barriers. Source: Zenuity.

3

Method

The methodology used in this thesis can be separated into three main parts; creation of the dataset, implementation of network models and evaluation of obtained results. The work flow has largely followed these steps, though some overlaps and iterations have been necessary. In the following sections, the data format, the generative networks and the evaluation are described in more detail.

An overview of setup for the project is shown in Figure 3.1. The part of the chart to the left of the dashed line, that is the driving scenario, the sensor readings and their respective tool chain of processing, are part of the toolchain at Zenuity and was not a part of this project. Here, a real life scenario was measured by both the lidar and the radar sensor. Both of the outputs were processed, so that the output from the radar was given at detection level and the output from the lidar at object level.

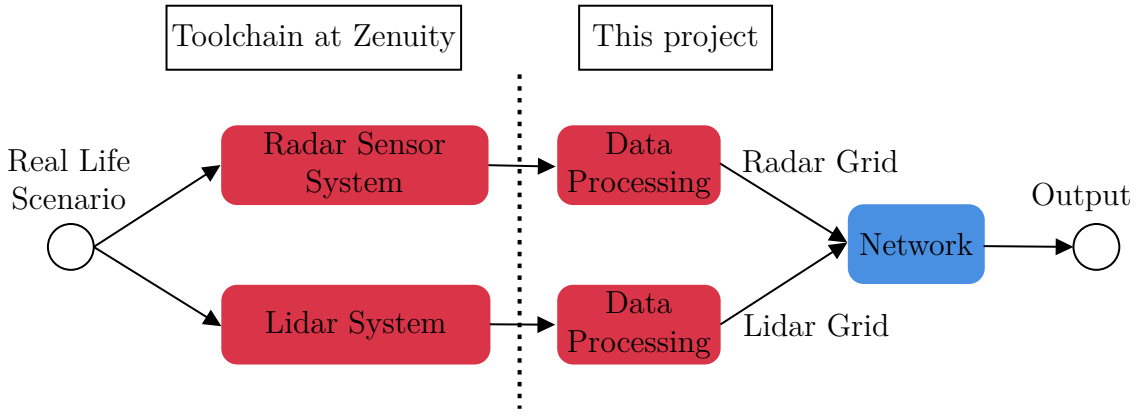


Figure 3.1: Overview of the setup for the thesis. The toolchain at Zenuity is briefly described to the left of the dashed line. To the right of the line, the steps that were implemented in this project are depicted.

The parts that were implemented for this project were the ones to the right of the dashed line. These consisted of a second data processing, where the sensor readings were transformed into a format suitable for neural network training, and the network itself. Both radar and lidar data was then used for training the network to produce

the desired output. When the network training was finished, the radar branch (that is the top row in Figure 3.1) was switched off and the network was to produce a radar output of its own.

3.1 Implementation and Hardware

The radar models were based on sensor data gathered by Zenuity. It consisted of automotive radar data from a front looking 77 GHz radar [41]. The front looking radar sensor had a view as described by Figure 3.2, i.e. it was smaller than 180° such that it was shaped like a fan. The sensor data also consisted of automotive lidar data from an HDL-64E Velodyne lidar with a range of 120 meters and 360° view [42].

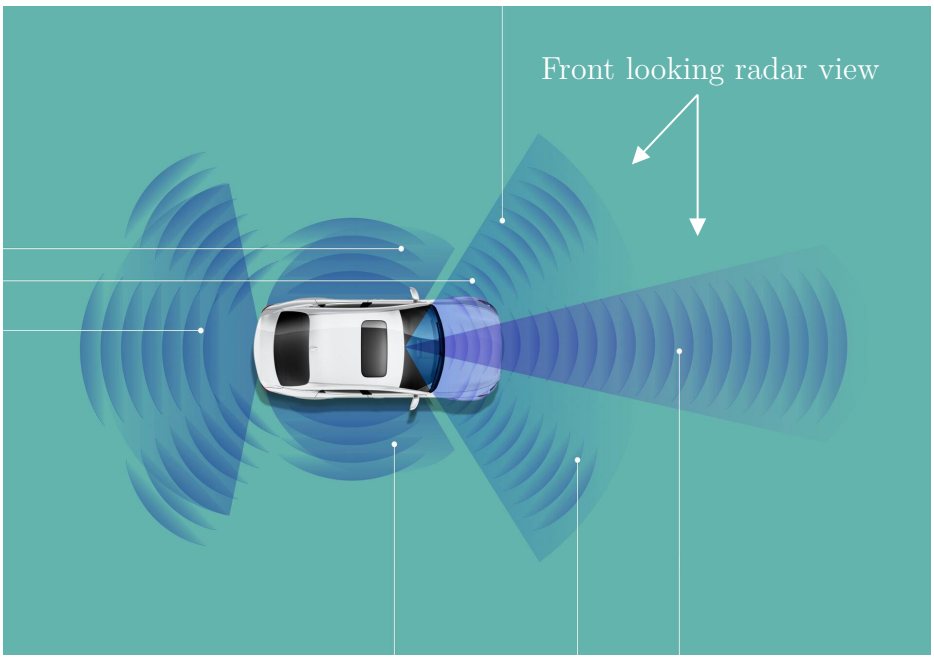


Figure 3.2: An illustration of the view of automotive radar sensors. The front looking radar sensor that was utilised to gather data had a view as described by the two overlapping circle segments that are in front of the vehicle. Source: Veoneer [41].

The radar models were created with Python, Tensorflow GPU and Keras. However, data processing for both dataset generation and model evaluation was performed in Matlab. A large part of the data processing that was performed for the dataset creation came from the Matlab code base of the Data Analysis Team at Zenuity. The hardware resouces available for the network training were an Intel Core i7-7820HQ CPU at 2.9 GHz with 4 Cores and a NVIDIA Quadro M2200 GPU.

3.2 Data

The data was collected by vehicles driven in highway environments under different conditions, in various countries. Only highway environments were included in this project, since they contain less noise than urban environments. The vehicles used to collect the data are equipped with e.g. radar and lidar as well as cameras that record the surroundings of the car continuously. The gathered data was then processed by algorithms in the tool chain at Zenuity. This resulted in a set of frames, for which one frame contains sensor readings for one measurement instance.

Radar data at detection level was used to train the networks. This data included point wise detections, with a lateral and longitudinal position for each detection. Several detections could exist for the same object. No tracking algorithm had been applied to this data. Hence, the detections were not tracked over time, why it was natural to train the networks on one frame at a time.

The lidar data constituted the conditional input to the models, and was given at object level. Thus, it was obtained with the help of object tracking, generating smooth object detections over time. It also included descriptions of the area of the objects and their types. The lidar had a lower sampling frequency than the radar sensor, why the lidar data was up sampled to match the radar data. The purpose of this data was to describe the surroundings of the vehicle to the models.

3.2.1 Coordinate System of the Data

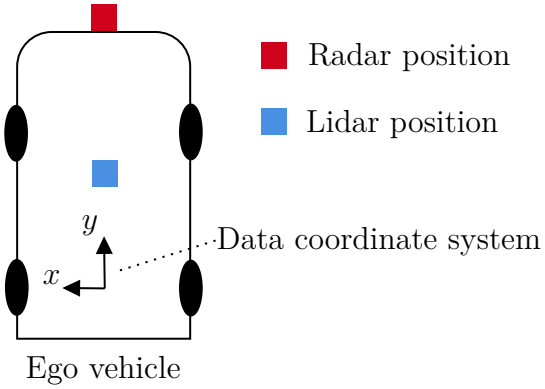


Figure 3.3: A description of the sensor positions on the ego vehicle. Both the radar and the lidar data has been formatted after the coordinate system described in the figure, such that origin is the rear axle of the car.

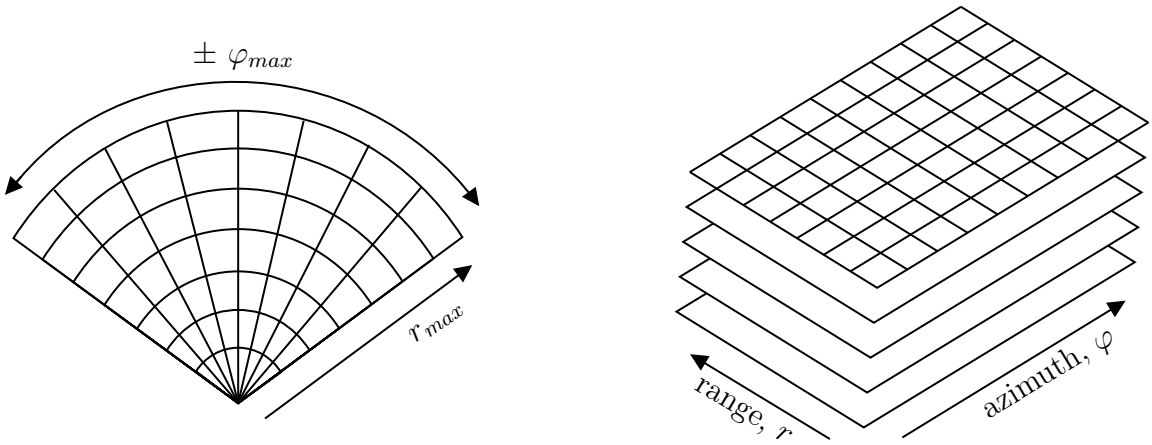
In Figure 3.3, the positions of the lidar and radar sensor on the vehicle are described. The radar sensor was placed on the front of the car, while the lidar was mounted

on the roof. This was adjusted for in the data processing at Zenuity, such that both data types had their origin at the rear axle of the ego vehicle.

Hence, the radar data had its origin behind the actual sensor. Since the radar sensor is front-looking, this resulted in an absence of detections in the radar data for positions behind the radar sensor, i.e. for positions with small longitudinal values. The lidar did not suffer from this, since it has a 360° view.

3.2.2 Data Format

Before training the models on the data, the readings from both the radar and lidar in each frame were formatted into binary matrices in polar coordinates, also called occupancy grids. Each position in a polar matrix represented a position in polar coordinates in the surroundings of the ego vehicle. See Figure 3.4a for a schematic view of what the grids looked like.



(a) Polar grid

(b) Grid formatted into a matrix, with several layers

Figure 3.4: Description of the polar grids used as input to the models. Each object and detection was placed in the corresponding location in the grid shown in a, where the ego vehicle is positioned at the apex. Hence, only objects and detections within a certain range and azimuth were considered. For the lidar grid, the different object classes were divided into separate grids, forming a three dimensional matrix as in b.

As a consequence of the grid format, the input data was limited to readings in a certain range and azimuth of the ego vehicle. The range of the polar matrix was set to $r_{\max}$ and the azimuth to $\pm\varphi_{\max}$. These limit choices were loosely based on the capacities of the radar and the lidar, and are left out of this report for confidentiality reasons. The grid contained 64 rows and 64 columns. A discussion pertaining the model restrictions and sensor capacities is provided in section 5.1.

In the occupancy grids, a value of one denotes that a radar detection or a lidar object is contained within that grid element. To describe an occupied spot in a grid, the term *grid point* is used. All ones in the grids were therefore denoted as grid points. Figure 3.5 and Figure 3.6 display an example environment in a frame and the corresponding set of grid points for both radar and lidar data.

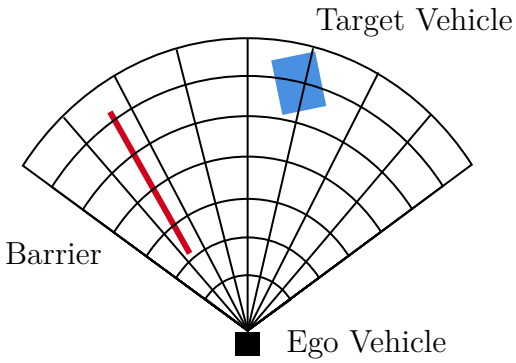
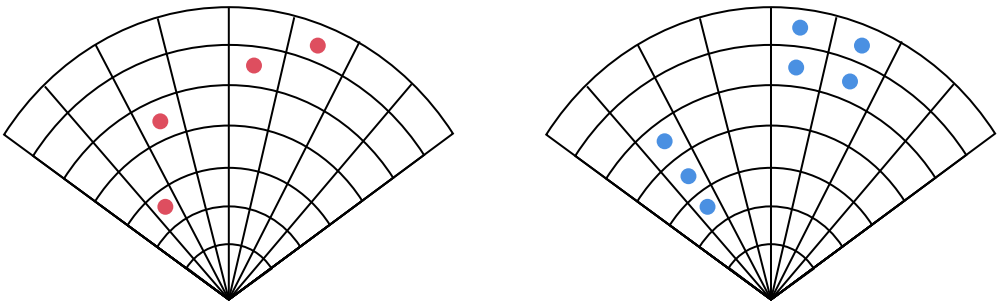


Figure 3.5: A toy example of what the environment around the ego vehicle may look like. Within the range of the grid, a barrier, in red, and another vehicle, in blue, are present. These are detected by the radar and the lidar sensors, processed and then transformed into the grid format, resulting in Figure 3.6.



(a) Radar Detections (b) Lidar Objects

Figure 3.6: Toy example of two grids, corresponding to the situation in Figure 3.5. In a, four grid points, corresponding to four radar detections, can be seen. These can belong to the same or different objects, as a radar detection pays no regard to measuring the extent of detected objects. In b, two objects are distinguishable by three respectively four grid points. The lidar data contains information about the bounding boxes of the objects. Hence, all locations in the grid that contain a bounding box will be marked for the lidar data, such that objects detected by the lidar in general have more grid points than the ones detected by the radar sensor. In this figure, no object class separation for the lidar objects is considered.

As the lidar data was given at object level, it contained information about the classes of the detected objects. To include this information in the data format,

several polar grids were stacked in layers with different classes in each layer. This is illustrated in Figure 3.4b. The classes were divided as follows, with each class in a separate grid layer: “miscellaneous”, “car, vehicle and truck”, “motorcycle and bicycle”, “buildings” and “barriers”.

Unlike the radar data, where each detection was given as a single position, the lidar objects also described the area of the objects. Hence, a radar detection resulted in only one grid point while a lidar object could result in many. See Figure 3.5 and Figure 3.6 for an example of the difference in nature between the radar detections and the lidar objects.

The aforementioned data formatting steps resulted in two sets of polar grids, one set for the radar and one for the lidar. Each set contained a number of grids that each corresponded to one frame in the dataset. An example of two polar grids that correspond to a frame in the dataset are shown in Figure 3.8. The colours used to indicate detections and different object classes for radar and lidar are shown in Figure 3.7.

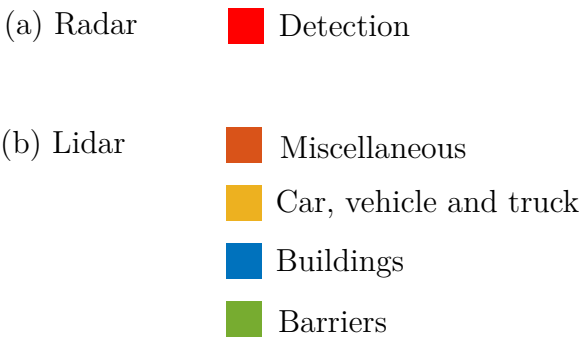
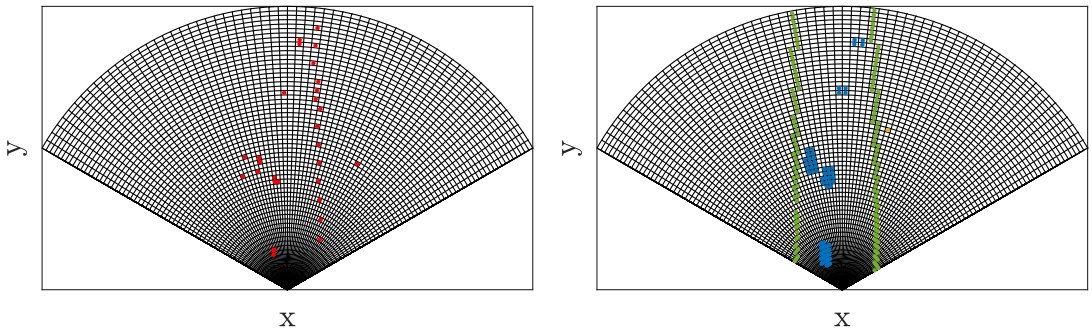


Figure 3.7: The colours used to describe detections and object classes in the polar grids. Since the radar detection level does not register object class only one colour is needed to describe the radar data. Hence, a red colour is used to indicate radar detections. As the lidar data also registers object class, several colours are needed to describe the data. Orange, yellow, blue and green are used to indicate different groupings of object classes for the lidar data.



(a) Radar detections. **(b)** Lidar objects.

Figure 3.8: Example of a frame from the dataset that was created for and used in this thesis. Figure 3.8a indicates the radar detections that were registered for the environment described by the lidar in Figure 3.8b. The lidar registers five different vehicles along with barriers on each side of the road, while the radar indicates a number of detections for both vehicles and barriers. The plot colours are explained in Figure 3.7.

3.2.3 Properties of the dataset

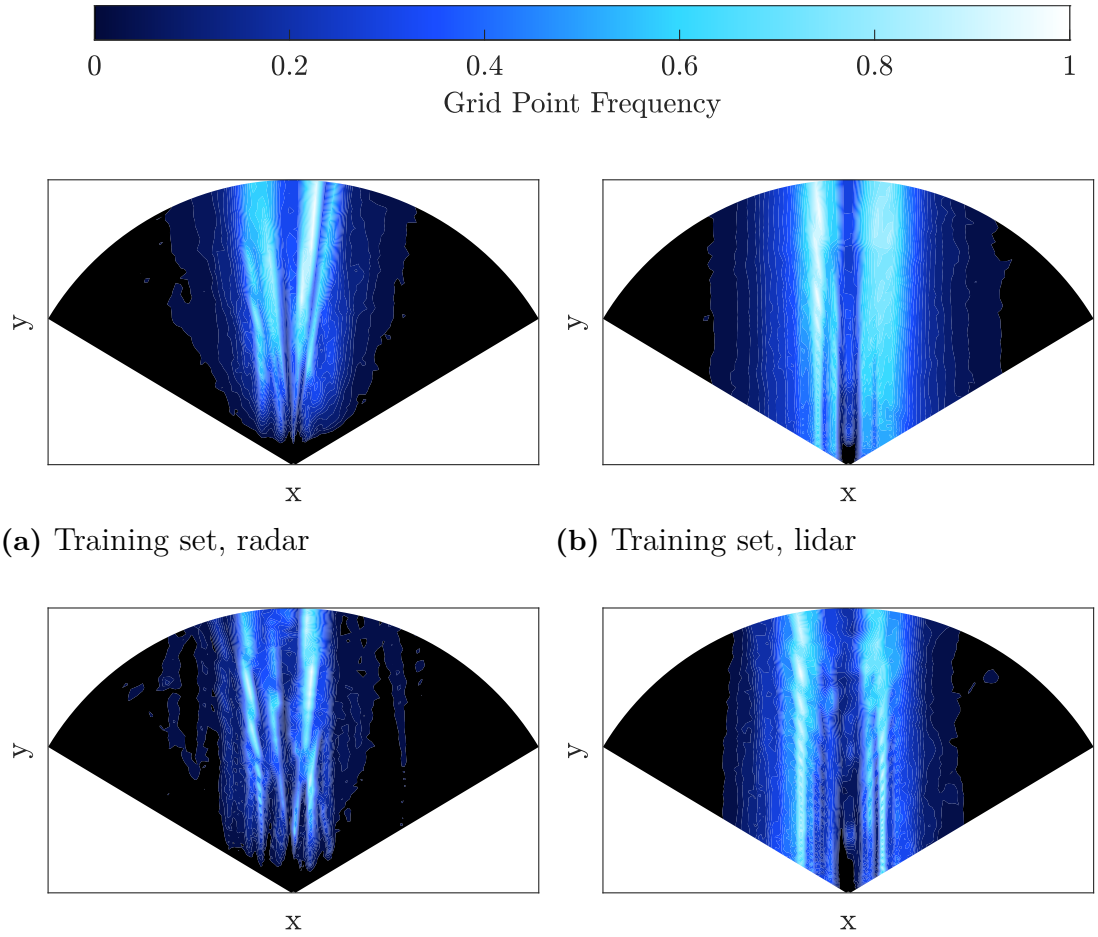
In this section, some of the properties of the created dataset are examined. The dataset consisted of 1,361,012 frames in total, all in the grid format described in section 3.2.2. Hence, each frame contained one radar grid and one lidar grid.

During training of the models, the data was divided into training and test sets. As the data consisted of consecutive frames, many of the frames in the datasets were almost identical. Therefore, it was important to use sufficiently large datasets when training the networks, and to shuffle the frames between training batches. For the same reason, it was also important to not shuffle the frames when dividing the data into training and test sets. Doing so would have resulted in near interchangeable frames appearing in both datasets, which would have undermined the original purpose of dividing the data into several sets.

The test set consisted of 14,342 frames, corresponding to 1% of the data. The radar models were not trained on this test data, so that it could be used to evaluate the generalisation capability of the models in Chapter 4. Below, the test set is compared to the training set, to argue for the representativeness of the test set. Since the frames were not shuffled when dividing the data, this analysis is necessary to ensure that the test set is not biased in some way, despite this.

In Figure 3.9, the frequencies of grid points throughout all frames in the datasets are depicted as heatmaps. The left column corresponds to radar data, and the right one to lidar data. The top row contains heatmaps over the training set, and the bottom row of the test set. To generate these, several grids such as the ones in Figure 3.8 were summed up. For the radar heatmaps in Figures 3.9a and 3.9c, grids corresponding to Figure 3.8a were summed, and for the lidar heatmaps in Figures 3.9b and 3.9d, grids corresponding to Figure 3.8b were summed, without respect to object class. The object classes in the lidar data were treated the same to make the comparison to the radar detections possible. The resulting grids have then been normalised, such that the largest value in each grid is 1. Hence, the scale is not the same in the figures, why only the general shape of the heatmaps should be considered, and not the actual colours.

The plots in Figures 3.9a and 3.9c showing the frequencies of the radar detections display several characteristics. Firstly, the detections are clearly concentrated to the highway in front of the ego vehicle, and even form a couple of lanes. Moreover, both heatmaps for the radar data display a grid point frequency of zero on the left and right edges of the grid and on short ranges. For example, note that the area directly in front of the ego vehicle, corresponding to a short range, is dark for both of the heatmaps in Figures 3.9a and 3.9c.



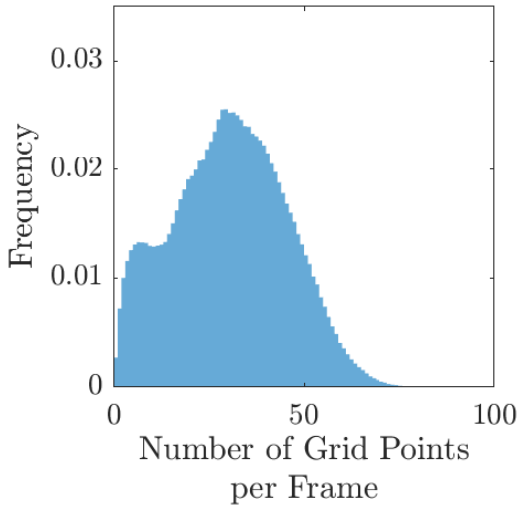
(a) Training set, radar

(b) Training set, lidar

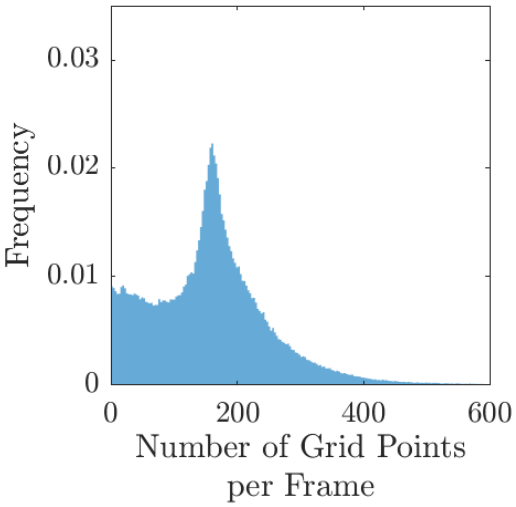
(c) Test set, radar

(d) Test set, lidar

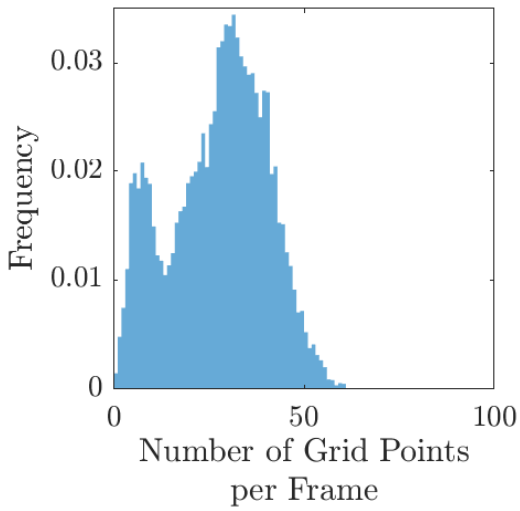
Figure 3.9: Heatmaps describing the frequency of grid points in each position for all frames in the datasets. Each figure is normalised to itself, why only the general shape of the heatmaps should be considered, and not the actual colours. The units are left out for confidentiality reasons.



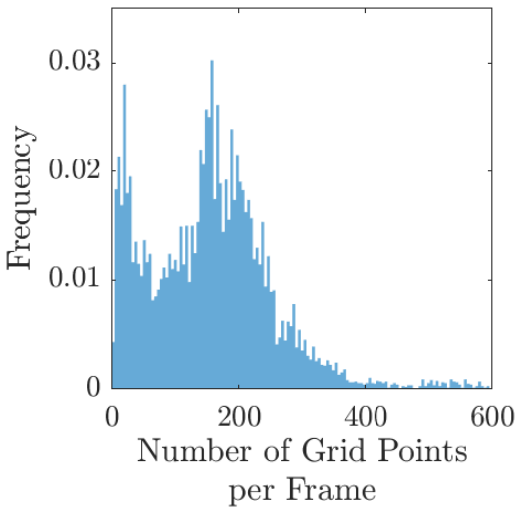
(a) Training set, radar



(b) Training set, lidar



(c) Test set, radar



(d) Test set, lidar

Figure 3.10: Histograms of the number of grid points per frame throughout the datasets. The frequencies are normalised. Note the difference in scale on the x-axis.

The plots in Figures 3.9b and 3.9d showing the frequency of the lidar objects also have certain properties. Similarly to the radar data, the lidar objects are contained within the lanes in front of the vehicle. As discussed in the section above, each lidar object is described as the set of grid points inside the object's bounding box. This is a part of the reason why the lanes in the lidar data heatmap are wider and smoother than for the radar data. Moreover, the lidar heatmaps differ from the radar versions on the left and right edges of the grid and at short ranges. They display a higher object frequency closer to the edges than the radar, and show no decrease in grid point frequency at short ranges. This is partly due to the shift of coordinate system described in section 3.2.1.

In Figure 3.10, the distribution of the number of grid points per frame can be seen. Histograms have been created for both the radar and the lidar, and for the training set as well as the test set. For lidar objects with bounding boxes, each grid square that is inside the bounding box is counted once, as seen in Figure 3.6b for the environment in Figure 3.5. This means that large objects will contribute more to the distribution, and that lidar objects generally have more grid points. This explains the difference in the scale in the x-axis in Figure 3.10 between the two sensors. Even though the scale is very different between the radar detections and the lidar objects, the distributions have resembling shapes.

Overall, Figures 3.9 and 3.10 show that although the radar detections and the lidar objects have some quantitative differences, the general qualitative behaviour is similar. Of course, the analysis given in this section is in no way an exhaustive comparison of the performance of the radar and lidar data. It only amounts to the general behaviour of the data, and no comparisons between single frames have been made. Nonetheless, the correspondence between the two data types indicates that the lidar data has the potential of being used as conditional input to the models in this project.

By comparing the test set with the training set in Figures 3.9 and 3.10, it is noted that while the plots for the test set are not as smooth as those for the training set, it still take on analogous properties. As seen by comparing Figure 3.10b and Figure 3.10d, the test set displays more variance, although it holds a general similarity to the rest of the data.

While the test set displays more variance, e.g. compare Figure 3.10d with Figure 3.10b, it holds a general similarity to the rest of the data. Given this analysis, the test set can be considered representative of the data, and can be used to evaluate models trained on the training set.

3.2.4 Test Frames

Ten test frames from the test set were used for the evaluation of the models, all of these may be viewed in appendix A. Four of these frames are used to illustrate some of the results in the next chapter. The input data for those frames, that is the radar grid and the lidar grid, along with a photo of the frame, are depicted in Figures 3.11 and 3.12. These test frames were chosen with the purpose of evaluating the radar models for different environments.

Test frame 1 is captured in a busy highway environment with several cars present, along with two barriers, see Figure 3.11a. Note that both of the barriers contain steel. Moreover, Figure 3.12b shows that the lidar data has captured the relevant objects in the photo. The radar data in Figure 3.12a also gives detections for both of the barriers as well as the cars. However, no detections are given for the barriers on a short range, close to the ego vehicle.

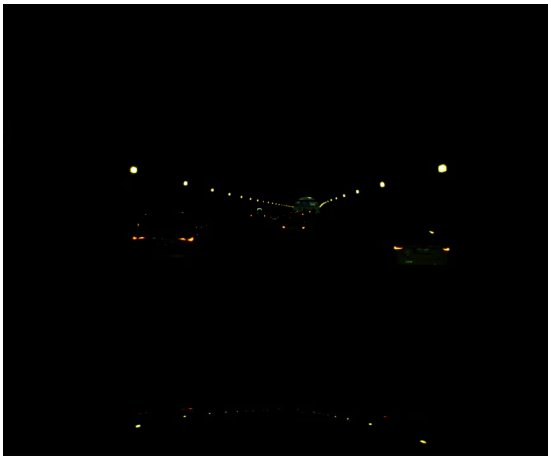
Test frame 2 is captured on a mostly lonely road with barriers on both sides, see Figure 3.11b. Here, the barrier to the left is made of steel while the one to the right is made of concrete. The lidar data in Figure 3.12d correctly registers both of the barriers closest to the ego vehicle. However, it shows a strange behaviour for the barrier to the left further away. Seemingly, the trees to the left of the road confuses the lidar such that it registers weirdly shaped *barrier noise* in the left part of the grid. The radar data in Figure 3.12c only gives detections for the steel barrier to the left, and completely misses the concrete barrier to the right. Note that the radar data does not contain any detections for the steel barrier on a closer range. Moreover, the radar data does not register anything in the region of the barrier noise.

Test frame 3 is captured in a dark tunnel with three other cars present, see Figure 3.11c. As can be seen in Figure 3.12f, the lidar data correctly registers the three cars along with the walls of the tunnel. The radar data in Figure 3.12e also gives detections for parts of the tunnel walls and for the three vehicles. Yet again, the radar data does not contain any detections on a short range.

Test frame 4 is captured on a road with a truck in close presence, along with a car further away and a concrete barrier to the right, see Figure 3.11d. As can be seen in Figure 3.12h the lidar data correctly registers the vehicles, while it completely misses the concrete barrier. The radar data in Figure 3.12g gives detections for both vehicles, while it displays none for the concrete barrier. Moreover, it generates extra detections to the left in the grid with no clear object correspondence with the photo in Figure 3.11d, as the truck obscures the view.

(a) Photo of test frame 1.

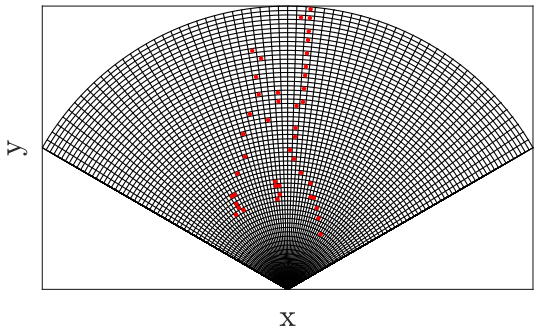
(b) Photo of test frame 2.



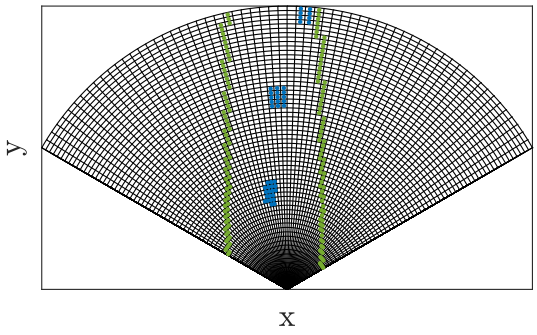
(c) Photo of test frame 3.

(d) Photo of test frame 4.

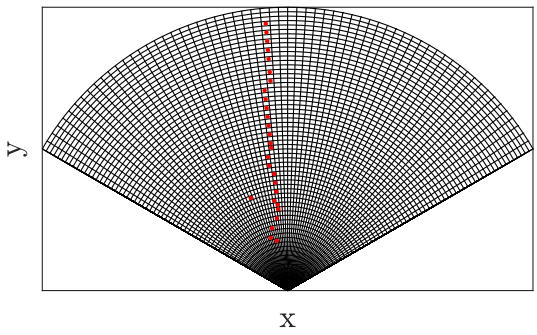
Figure 3.11: Photos of test frames 1, 2, 3 and 4, taken from the ego vehicle. The evaluation of the radar models will include an analysis of the model performance on these frames.



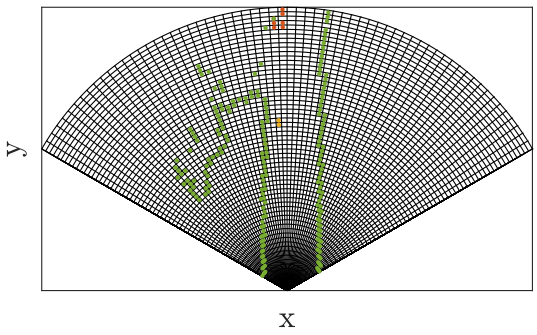
(a) Test frame 1, radar detections.



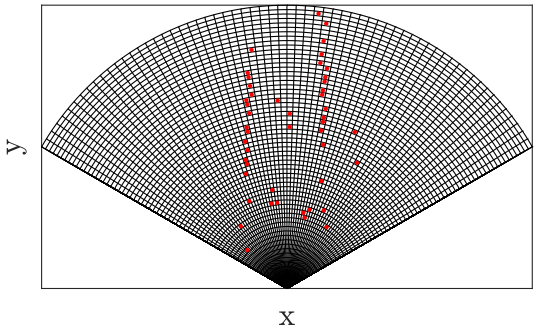
(b) Test frame 1, lidar objects.



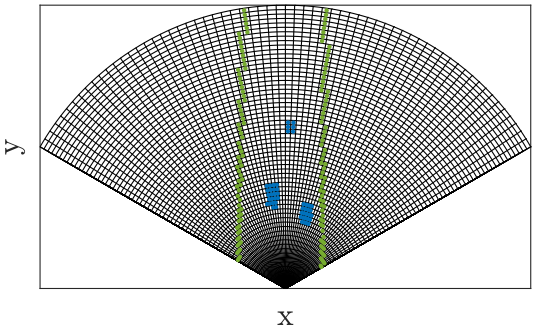
(c) Test frame 2, radar detections.



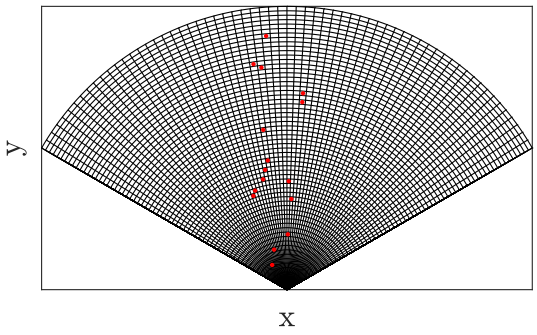
(d) Test frame 2, lidar objects.



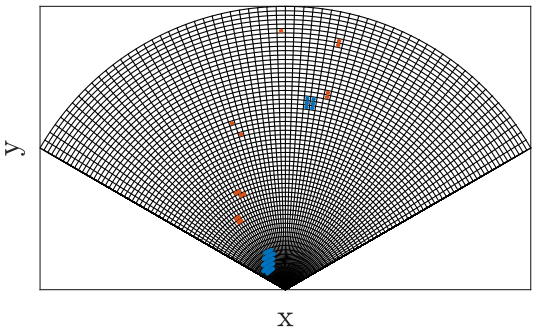
(e) Test frame 3, radar detections.



(f) Test frame 3, lidar objects.



(g) Test frame 4, radar detections.



(h) Test frame 4, lidar objects.

Figure 3.12: The radar and lidar grid for the test frames 1, 2, 3 and 4. The colours correspond to the labels in Figure 3.7. See Figure 3.11 for photos of the scenarios. Note that the radar in (c) misses the barrier that the lidar detects to the right in (d), since it is made of concrete. Moreover, in (d) the lidar erroneously registers a barrier further away to the left. In (h), note that the lidar also misses the concrete barrier to the right, seen in Figure 3.11d.

3.3 Models

The network models implemented in this project were largely inspired by the network architectures in [5]. Various versions of VAEs and GANs were implemented, along with combinations of these. Furthermore, different loss functions, hyperparameters and training procedures were applied to these. The following sections contain descriptions of the networks that were implemented. To facilitate the description of the network implementation, the notation in Figure 3.13 and Figure 3.14 is used.

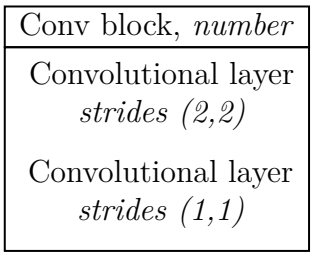


Figure 3.13: To compress the description of the network implementation, the notation Conv block, *number* is used. A convolutional block here consists of two convolutional layers. The first convolutional layer has strides (2,2) whereas the second one has strides (1,1). Both layers have kernel size (3,3). *number* denotes the number of filters in the convolutional layers. In the cases for which a second digit is given inside parentheses, this corresponds to the second convolutional layer. The activation function that is used for each layer is either the rectified linear unit (ReLu) or the leaky rectified linear unit (lReLu). Batch normalization may also be implemented for the layers in this block. Conv^T block, *number* works in a similar fashion, but with transposed convolutional layers.

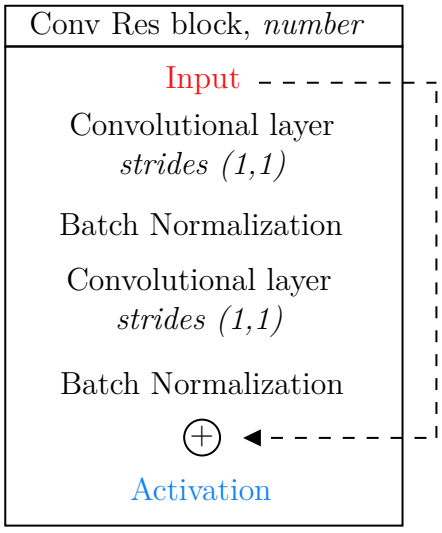


Figure 3.14: To compress the description of the network implementation, the notation Conv Res block, *number* is used to describe a convolutional residual block that is used for the models. The block consists of two convolutional layers with batch normalisation in between. The first convolutional layer has strides (1,1) whereas the second one has strides (1,1). Both layers have kernel size (3,3). *number* denotes the number of filters in the convolutional layers. In the cases where a second digit is given inside a parenthesis, it corresponds to the second convolutional layer. The activation function that is used for this block is the rectified linear unit (ReLu). Batch normalization is performed either before or after the activation function of the layers, as will be specified for each model.

3.3.1 Implementation of Conditional Variational Autoencoders

A schematic view of the CVAE network that was implemented can be seen in Figure 3.15. As described in sections 2.2.1 and 2.2.2, the loss of the network is a combination of the KL-loss and the CVAE-loss. An important part of implementing a CVAE was to define a loss function that was suitable for its task. The loss function used in this project is described further below.

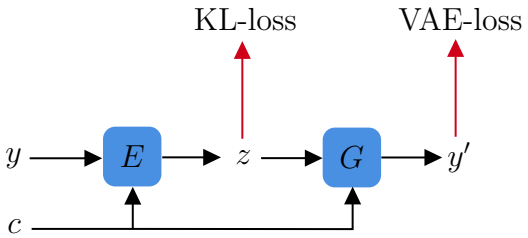


Figure 3.15: The implementation of the conditional variational autoencoder network. E denotes the encoding layers and G the generating layers. The sampling and generative process is the same as in Figure 2.8 and Figure 2.9.

As mentioned in Chapter 2, the goal of a VAE is to maximise Equation (2.1). Hence, the loss function of a VAE consists of two terms, namely the Kullback Leibler-divergence l_{KL} defining the deviation of the latent variable z from a normal distribution, and a log-likelihood term l_G representing the generator loss. The total loss is then computed as $l = w_G l_G - w_{KL} l_{KL}$, where w_G and w_{KL} are tunable weights. The KL-loss is defined as

$$l_{KL} = \frac{1}{d_z} \sum_{i=1}^{d_z} 1 + z_{\sigma,i} - z_{\mu,i}^2 - e^{z_{\sigma,i}}.$$

Here, z_σ and z_μ are latent parameters, learned by the model. d_z is the dimension of the latent code, z, in the network.

The log-likelihood term is defined as

$$l_G = \log(logP_\sigma) + \frac{error}{logP_\sigma},$$

where $logP_\sigma$ is a scale parameter and $error$ is defined in the generator loss function. There are many ways of defining a loss function for the generator in a VAE. This is a crucial choice since the loss function, in combination with the KL-loss, will direct the training of the network and define the behaviour of the output.

A dilemma in this project was that the training data was very sparse. Most positions in the grids were unoccupied, which resulted in an imbalanced dataset. Using a standard loss function, for example the mean squared error between the generated

output and the true radar detections, often resulted in a network that was very hesitant to generate detections, since they were so unlikely. To circumvent this, a loss function that treats the zeroes and ones in the grids separately was needed for the generator loss. Here, the loss function proposed by Wang et al. in [43], called *mean false error* (MFE), was used. It is designed to improve the performance when training on imbalanced datsets, by being more sensitive to errors in the minority class. The loss function is defined as

$$MFE = c_1 \cdot FPE + c_2 \cdot FNE,$$

where

$$FPE = \frac{1}{N} \sum_{i=1}^{N} \left(y'^{(1)}_i - y^{(1)}_i \right)^2,$$

$$FNE = \frac{1}{P} \sum_{i=1}^{P} \left(y'^{(0)}_i - y^{(0)}_i \right)^2.$$

FPE and FNE are the mean false positive error and mean false negative error, respectively. N and P are the numbers of samples in each class. The class is either zero or one, denoting whether a grid point should exist in the binary positional radar grid, consisting of $N+P$ elements. y' is the generated output and y is the true radar data. The loss function can be adjusted by weighting FPE and FNE using c_1 and c_2.

A detailed description of the network implementation is given in Figure 3.16. In Table 3.1, the hyperparameters used in the training are listed. The hyperparameters were chosen to ensure a stable training process without overfitting. The model contained 166,277 trainable parameters and was trained on the dataset for 30 epochs during approximately 15 hours. The training was stopped when the network accuracy started to converge to a constant value for a validation dataset.

Table 3.1: The hyperparameters used for training the CVAE. The parameters were tuned to improve upon the training of the network described in Figure 3.16.

Hyperparameter	Description	Value
b	Batch size.	128
n_{epoch}	Number of epochs.	30
l_2	L2-regularisation parameter.	0.001
$logP_\sigma$	Scale parameter for the log-likelihood.	0.001
w_G	Weight for the generator loss function.	0.5
w_{KL}	Weight for the KL-loss.	0.5
c_1	Loss function weight for ones.	1
c_2	Loss function weight for zeroes.	3
σ_ε	Variance for sampling layer.	0.01

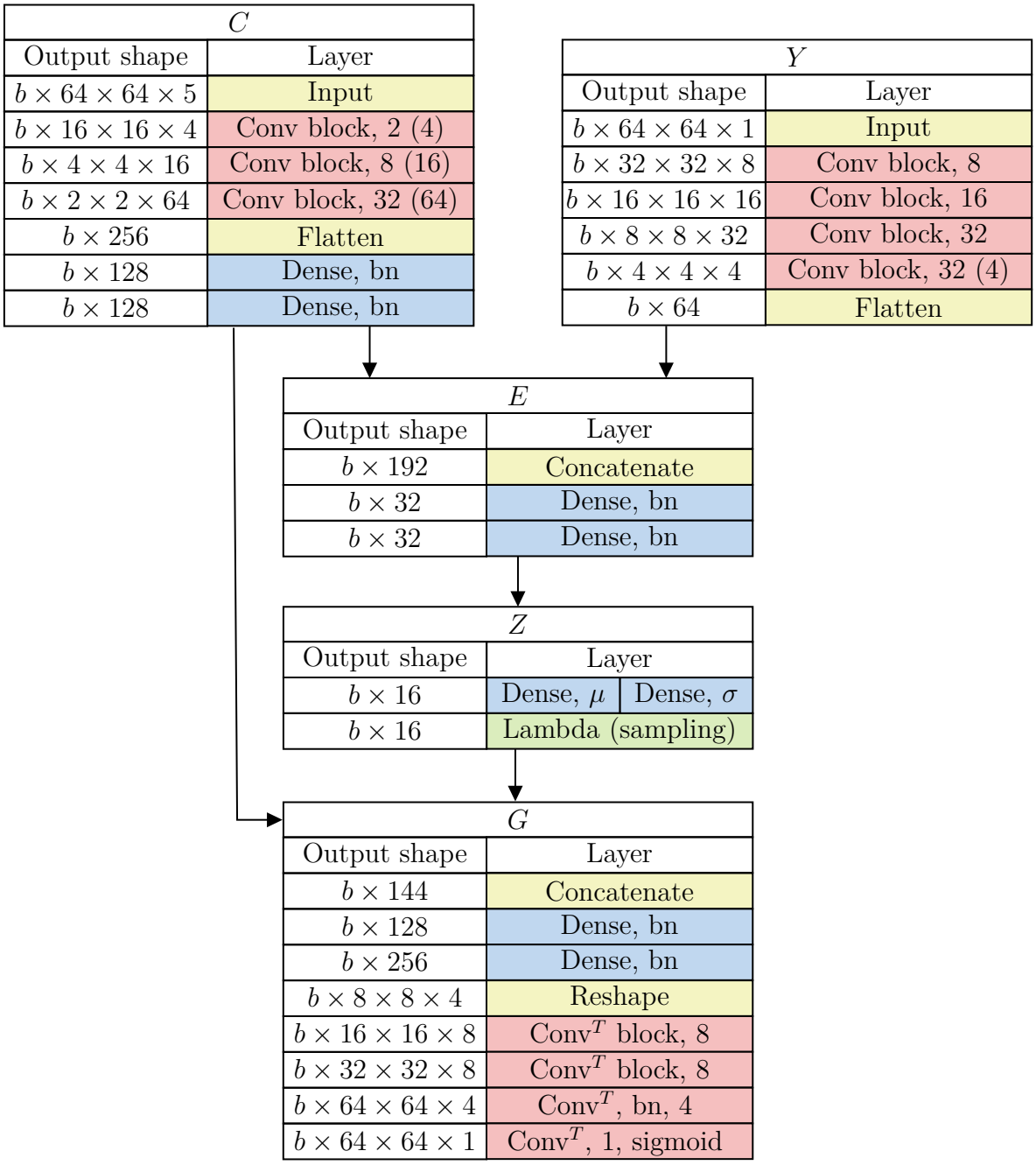


Figure 3.16: Description of the implementation of the CVAE. The figure follows the notation described in Figure 3.13. "Dense, bn" denotes a dense layer followed by batch normalization. The radar data is given as input to Y and the conditional input (the lidar data) is fed to C. These branches are then concatenated in the encoder. In Z, the learned μ and σ are used for sampling in the Lambda layer. The output from Z, z, is given by $z = \mu + \varepsilon\sigma$, where $\varepsilon \sim N(0, \sigma_\varepsilon)$. When the training was done, the layers in C and in the generator were kept to obtain a generating model. For the generating model, the input to the generator is drawn from $\sim N(0, I)$, where I is the identity matrix, instead of from the sampling in Z.

3.3.2 Implementation of Conditional Generative Adversarial Networks

Conditional GANs were also implemented to model the radar sensor. More specifically, conditional deep convolutional generative adversarial networks (cDCGANs) were used, with a general structure as described by Figure 3.17. Note that the generator did not have any noise z as input, such that generated samples only were based on the conditional input c.

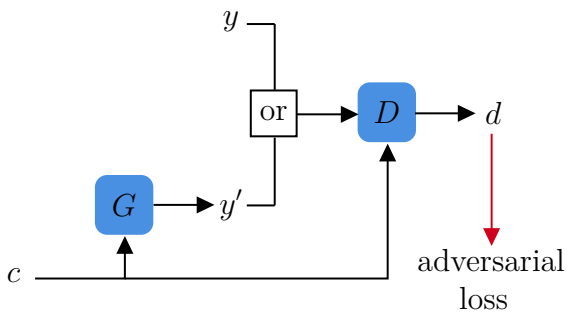


Figure 3.17: The implementation of the cDCGAN. For each data frame the generator G receives a conditional input, c. G then generates a fake data frame, y', that is fed to the discriminator D along with c. D then outputs a probability d of the received sample being true or false. G is then trained on the loss based on this d for the generated y'. D is trained on both generated samples, y', and true samples, y.

A detailed description of the layers in the generator and the discriminator for the final GAN model version can be seen in Figure 3.18. The generator for this model consisted of 11 layers and was solely based on a residual block architecture, as illustrated in Figure 2.1. This was easily implemented as the input and output dimensions for G were of the same grid format, 64×64, as seen in Figure 3.18a. The discriminator for the model consisted of 20 layers, of which the majority were convolutional layers. Since D took both a conditional input and a radar detection input it utilised parallel layers for separate input processing.

A training algorithm, that described how the discriminator and generator were trained simultaneously, was needed for the training of a GAN. As described in section 2.3.2, the training process of a DCGAN is not necessarily simple, and there are some methods to stabilise it. For this thesis, a training algorithm that was based on the methods described in section 2.3.2 was derived for the training of the network in Figures 3.18a and 3.18b.

The derived algorithm used for the GAN training is described in Algorithm 1. Essentially, the algorithm describes that the decoder should be trained only if its accuracy on true or generated frames drops below a certain threshold, and that the generator should be trained otherwise. If a plateau in training is reached, the learning rate of both generator and discriminator is decreased and the discriminator is trained

G	
Output shape	Layer
$b \times 64 \times 64 \times 5$	Input, bn
$b \times 64 \times 64 \times 32$	Conv, 32, bn
$b \times 64 \times 64 \times 32$	Conv Res block, 32
	. . .
$b \times 64 \times 64 \times 1$	Conv, 1, sigmoid

(a) Description of the implementation of the generator, G, of the cDCGAN, following the notation described in Figure 3.14. Eight such blocks were used for the generator, denoted "Conv Res block, 32". ReLus are used where nothing else is specified, and batch normalization is performed before the activation function of each layer. No batch normalization is performed on the output. The input to the generator consists of the conditional input, that is the lidar data. The generator output is obtained after activation with a sigmoid function. In total, the generator network consisted of 150,827 trainable parameters.

C	
Output shape	Layer
$b \times 64 \times 64 \times 5$	Input
$b \times 32 \times 32 \times 4$	Conv, 4, bn
$b \times 16 \times 16 \times 8$	Conv, 8, bn
$b \times 8 \times 8 \times 16$	Conv, 16, bn
$b \times 4 \times 4 \times 32$	Conv, 32, bn
$b \times 2 \times 2 \times 64$	Conv, 64, bn
$b \times 1 \times 1 \times 128$	Conv, 128, bn
$b \times 128$	Flatten
$b \times 128$	Dense, bn
$b \times 128$	Dense, bn

Y	
Output shape	Layer
$b \times 64 \times 64 \times 1$	Input
$b \times 32 \times 32 \times 8$	Conv block, 8, bn
$b \times 16 \times 16 \times 16$	Conv block, 16, bn
$b \times 8 \times 8 \times 32$	Conv block, 32, bn
$b \times 4 \times 4 \times 4$	Conv block, 32 (4), bn
$b \times 64$	Flatten

D	
Output shape	Layer
$b \times 192$	Concatenate
$b \times 64$	Dense, bn
$b \times 32$	Dense, bn
$b \times 1$	Dense, sigmoid

(b) Description of the implementation of the discriminator in the cDCGAN. The conditional input of the discriminator was given to C and the radar data, either true or generated, was given to Y. The output from C and Y is then concatenated in D which generates the final output. C, Y and D together forms the discriminator. The figure follows the notation described in figure 3.13 for the Conv blocks. The convolutional layers in C have a stride of (2,2). The Leaky ReLU activation function was used for all layers in the discriminator, unless otherwise specified, and batch normalization was performed after the activation function for each layer. However, no batch normalization was performed on the inputs. In total, the discriminator network consisted of 176,077 trainable parameters.

Figure 3.18: Implementation of the cDCGAN.

if it is not too close to saturation. For each training step of either generator or discriminator, the network is trained on on a number of batches from the dataset, rather than the whole training set (corresponding to 1 epoch). Generally, the generator was trained on more batches for one step than the discriminator, since the generator was slower to learn. This process was then reiterated for a desired number of training steps.

Algorithm 1 GAN Training Algorithm

for i=1:n_{train} **do**
  if train_D **or** ((D_val_accuracy_on_true<$t_{D,\text{true}}$
    or D_val_accuracy_on_false<$t_{D,\text{false}}$)
    and (G_val_accuracy>$t_{G,\text{bottom}}$
    or D_val_accuracy_on_true<$t_{D,\text{true,bottom}}$)) **then**
    Train D for $n_{D,\text{train}}$ steps.
    train_D = False
    step_after_D_train = 0
    {Generally, D is trained for fewer steps than G.}
  else
    Train G for $n_{G,\text{train}}$ steps.
    step_after_D_train += 1
  end if
  Validate D and G.
  Save D_val_accuracy_on_true.
  Save D_val_accuracy_on_false.
  Save G_val_accuracy.
  if step_after_D_train $\geq n_{\text{min,plateau}}$ **then**
    Get average_diff for G_val_accuracy from past $n_{\text{min,plateau}}$ steps.
    if average_diff $< t_{\text{plateau}}$ **then**
      Decrease D learning rate by multiplying with α_D.
      Decrease G learning rate by multiplying with α_G.
      if (G_val_accuracy>$t_{G,\text{bottom}}$
        or D_val_accuracy_on_true<$t_{D,\text{true,bottom}}$) **then**
        train_D = True
      end if
    end if
  end if
end for

The hyperparameters for generator and discriminator were set as described in Table 3.2. The main goal of the parameter settings was to prevent either D or G from saturating. The optimiser for both D and G was the Adam algorithm [44], with initial learning rates given by $\gamma_{D,\text{init}}$ and $\gamma_{G,\text{init}}$ respectively. The β_1 value of Adam was also tuned for each network, as seen in Table 3.2. With the chosen training algorithm and parameters, the GAN model described by Figures 3.18a and 3.18b was trained on approximately 5,120,000 data frames, with some repetitions, for approximately 24 hours. The training was stopped when the network accuracy started to

converge to a constant value for a validation dataset. No convergence was attained for the GAN model, in the sense that G never reached a point at which D no longer could be trained to distinguish between generated and real samples.

Table 3.2: The hyperparameters used for training a GAN with Algorithm 1. All of the parameters were tuned for the training of the network described by Figures 3.18a and 3.18b.

Hyperparameter	Description	Value
b	Batch size.	64
n_{train}	Number of training steps for the model.	400
$n_{D,\text{train}}$	Number of training steps for D.	5
$n_{G,\text{train}}$	Number of training steps for G.	200
$n_{\text{min,plateau}}$	Minimum number of training steps in a plateau.	4
t_{plateau}	Threshold in difference of accuracy for a plateau.	0.02
$t_{D,\text{true}}$	Threshold in accuracy on y data for D.	0.7
$t_{D,\text{false}}$	Threshold in accuracy on y' data for D.	0.6
$t_{G,\text{bottom}}$	Bottom threshold in accuracy for G.	0.2
$t_{D,\text{true,bottom}}$	Bottom threshold in accuracy on y data for D.	0.6
$\gamma_{D,\text{init}}$	Initial learning rate for D.	0.0001
$\gamma_{G,\text{init}}$	Initial learning rate for G.	0.00001
$\beta_{1,D}$	β_1 value for D.	0.5
$\beta_{1,G}$	β_1 value for G.	0.99
α_D	Decrease in learning rate for D.	0.9
α_G	Decrease in learning rate for G.	0.9

3.4 Detection Sampling from Network Output

The networks were trained on binary data, as described in section 3.2. However, the networks can never output something that is completely binary. It will always be beneficial for them to include some uncertainty in the output. Hence, the output from the networks were matrices with values ranging from 0 to 1. This output can be interpreted as a probability distribution, where a grid element with a value close to 1 indicates that it is likely to find a detection in the area covered by the grid element.

Since the output from the models were probability distributions, a sampling had to be performed to get separate radar detections. The sampling can be performed in many ways, preferably with parameters corresponding to the dataset. No significant effort was put into deriving a sampling method for the radar models in this thesis. However, for illustrative and pedagogical reasons, some sampled frames are included in the results. For these, a naive sampling method was used, where detections were sampled from the radar detection distribution $P_{\text{detection}}(x)$ for a position x with an added constant threshold, τ. The distribution that was used to generate samples

can thus be described as,

$$P^{\tau}_{\text{detection}}(x) = P_{\text{detection}}(x) - \tau. \tag{3.1}$$

Consequently, the threshold adjusts the number of expected detections for each frame. A higher threshold leads to a lower number of expected detections from $P^{\tau}_{\text{detection}}(x)$.

3.5 Evaluation Methods

A stochastic generative radar model has no straightforward evaluation method. Since the aim of this project was to create a stochastic generative radar model, the evaluation of the results was not straightforward. On one hand, the output from the model should be similar to the true radar detections used as labels in the networks, as well as the the conditional input. Here, *similar* does not necessarily refer to identical, but rather that the generated output resembles the true one, with some variation. The generated detections should also correspond to the conditional input for each frame. On the other hand, the model should display the general behaviour of a radar. For example, it should on average generate as many detections as a real radar sensor would, and in the same ranges.

Thus, evaluating and comparing different models was not an easy task, and many different aspects had to be considered. Therefore, several different approaches were used in this project when testing the different radar models. These mainly concerned the average behaviour of the model and frame by frame comparisons.

As already mentioned, the output from the model should on average display the same behaviour as a real radar sensor would. Hence, comparisons of the distributions of different features over all frames were included in the evaluation. Histograms and heatmaps like the ones in section 3.2.3 were computed for the generated detections corresponding to the test set and then compared.

The generated output was also compared frame by frame to the real radar detections and conditional input. Ten frames from the test set were chosen to be representative of the data, and these frames were used to evaluate the results from all models. Four of these frames were described in section 3.2.4, and they will be used in Chapter 4 to describe the model results for some specific scenarios.

The model should also be stochastic. That is, the output from the model should be different every time the model is used. Since the output from a model was interpreted as a probability distribution, this was always accounted for in the sampling. The stochasticity was also ensured in the models that were given random noise as input.

To conclude, visual evaluation was the primary tool in evaluating the results of the implemented radar models. Similarly to previous work on radar modelling, modelled

radar detections were visually compared to true radar detections [8]. Because of the complex behaviour of a radar sensor, most loss functions that may be implemented to get a model accuracy metric are not suitable, and visual evaluation turns out to be the best option.

4

Result

When training the models presented in section 3.3 with the data described in section 3.2, both of them show potential of being useful for radar modelling. Even though both the VAE and GAN are deep generative models that share many properties, they exhibit several differences in their generated output. Both models have their own advantages as well as difficulties.

In general, the networks do not produce a binary output. Hence, the output is interpreted as a probability distribution, describing the probability of detections for each position in each frame. The sampling method described in section 3.4 is used to generate detections from this distribution. This makes it possible to compare detections from the radar model with those of a real radar sensor.

Both models are trained on the dataset described in section 3.2.3. They are then tested on the test set described in the same section. The output from the models is visualised in the same way as the dataset, using heatmaps and histograms. The output corresponding to the four test frames presented in section 3.2.4 is also included, frame by frame. In the following sections, the results from the VAE and GAN models described in Chapter 3 are presented further.

4.1 Predictions from the Conditional Variational Autoencoder

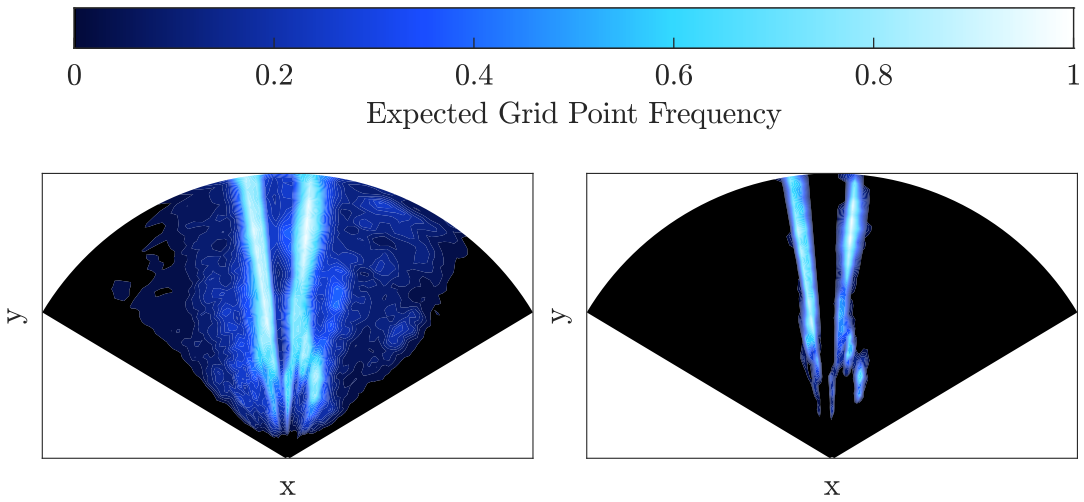
In this section, the result from the VAE implementation described in section 3.3.1 is presented. The generated samples are based on the conditional input, that is the lidar data, from the test set. The overall performance on the test set is visualised in Figures 4.1 and 4.2, and separate outputs from the four test frames described in section 3.2.4 are displayed in Figure 4.3.

In Figure 4.1a, the predicted output for all frames in the test set are depicted as a heatmap. All of the generated samples were added up to produce the heatmap, before performing any sampling. Compare with the real data in the test set in

Figure 3.9c. Looking at the two white lanes in the middle of the grid in Figure 4.1a, and the dark borders, it is clear that the model is able to learn in which areas detections are more likely. Furthermore, the lanes in Figure 4.1a are very similar to the ones for the whole dataset in Figure 3.9a, more so than to the test set in Figure 3.9c. However, the grid point frequency around the lanes is a bit more smudged than for both Figure 3.9c and Figure 3.9a.

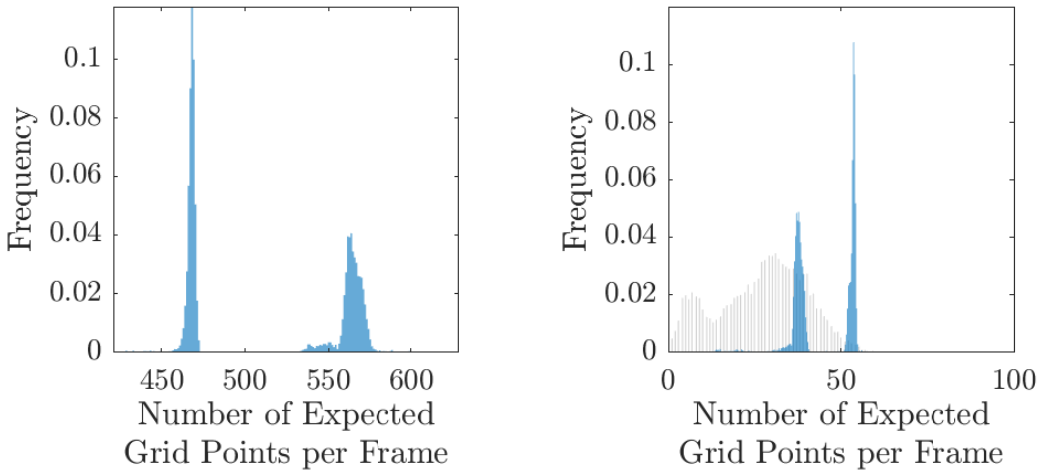
In Figure 4.1b, the predicted output for all frames in the test set has been subtracted with a threshold of $\tau = 0.4$, after which the output has been averaged over to create a heatmap. The resulting distribution limits the likelihood of detection to two lane shaped regions in front of the ego vehicle, and there is no detection probability for the regions that are not in the centre of the polar grid. As a result, it is not very similar to the average behaviour of the radar data in figures 3.9a and 3.9c, as the radar data also exhibits a probability of detection in the grid margins.

The histogram in Figure 4.2a clearly shows that this model is much more prone to generate detections than a real radar sensor. This is clear when comparing with Figure 3.10c. It does, however, exhibit peaks similar to the ones in the test set, although more distinct and at different values.



(a) Output from network. **(b)** After thresholding.

Figure 4.1: Heatmap over predicted grid points, i.e. predicted radar detections, from the VAE model. Predictions for all frames in the test set have been added to generate the heatmap. In a, the raw output from the network is depicted. In b, a threshold of 0.4 has been subtracted before adding the frames. The result can be compared to the true data in Figure 3.9c. Note that since the output from the model is not binary, but a number between 0 and 1, it is the expected number of detections that is depicted.



(a) Output from network. **(b)** After thresholding.

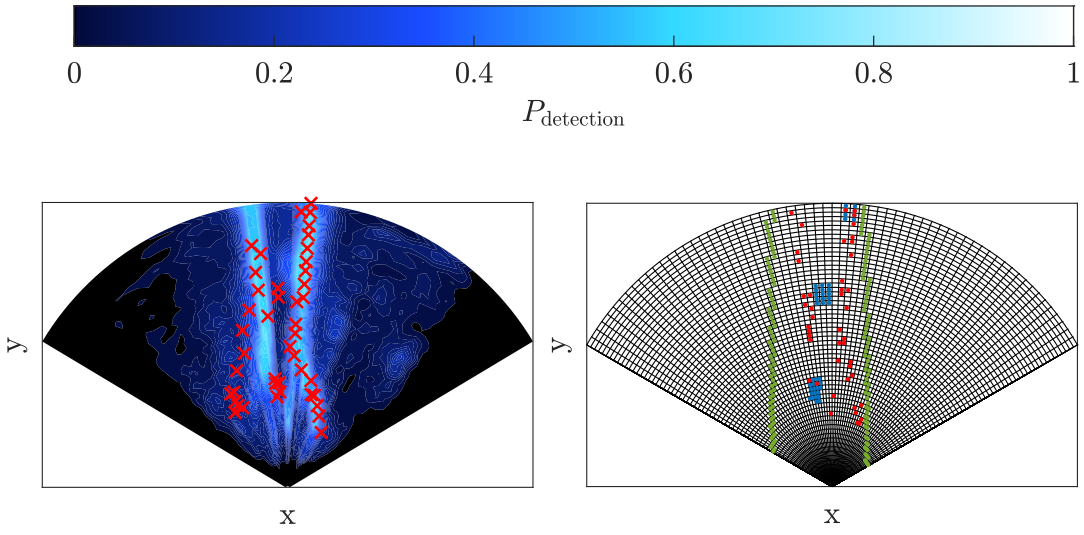
Figure 4.2: Histograms over the number of detections per frame, as predicted by the VAE model, using all test frames in the data set.Note that since the output from the model is not binary, but a number between 0 and 1, it is the expected number of detections that is depicted in these figures. Figure 4.2a depicts the distribution from the raw output from the network, before any sampling has been performed. In Figure 4.2b, a threshold of 0.4 has been subtracted from all probabilities before computing the expected number of detections. The true number of grid points per frame, as was depicted in Figure 3.10c, is included in grey in the background.

In Figure 4.2b, the distribution of the expected number of frames after sampling with Equation (3.1) is visualised. Here, a threshold of $\tau = 0.4$ was applied before the expected number of detections per frame was computed. This results in values in the same range as the true data in Figure 3.10c. However, even after thresholding, the shape of distribution for the VAE in Figure 4.2b is not similar to that of the distribution for the radar data in Figure 3.10c.

In Figure 4.3, the generated outputs for the four test frames described in section 3.2.4 are depicted. The left column contains heatmaps describing the output from the network, ranging from 0 to 1, for each frame. The red crosses are the true radar detections, included for comparison. The network produces little variation between the four frames, and it is hard to find a clear correspondence with the conditional input.

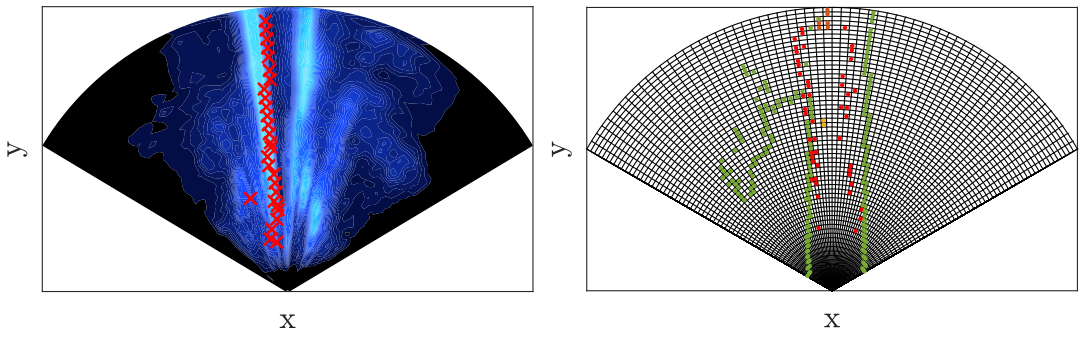
In the right column of Figure 4.3, detections have been sampled by interpreting the output as a probability distribution and using the naive sampling method described in section 3.4, with a threshold of 0.4. The threshold was chosen to decrease the number of sampled detections, and to make them similar to the true ones. The sampled detections are depicted in red. To illustrate the correspondence between sampled output and conditional input, the conditional input from the lidar has also been included for the plots in the right column. The plot colours are explained in Figure 3.7. Visibly, there is little correspondence between the sampled output and

the conditional input for the VAE.



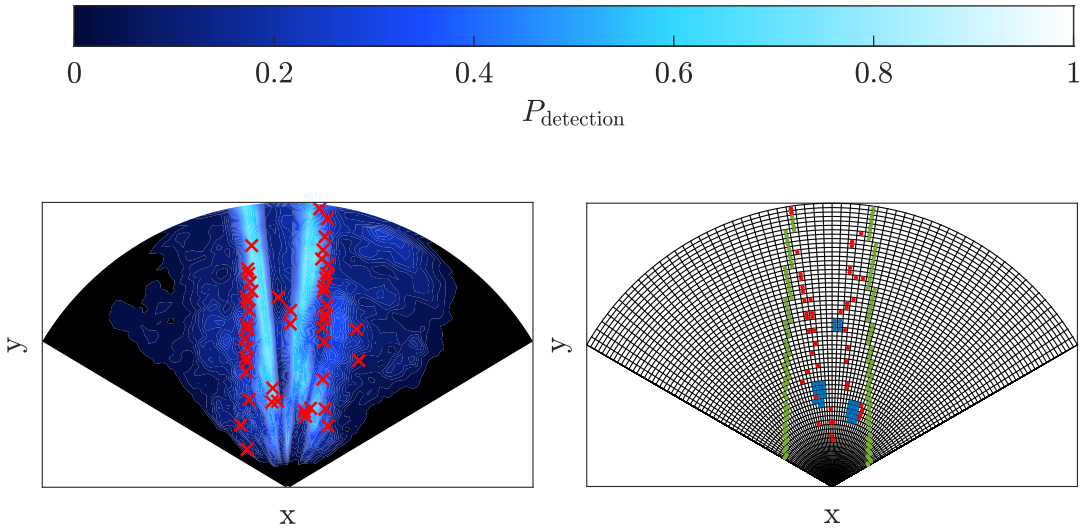
(a) Output, test frame 1.

(b) Sampling, test frame 1.



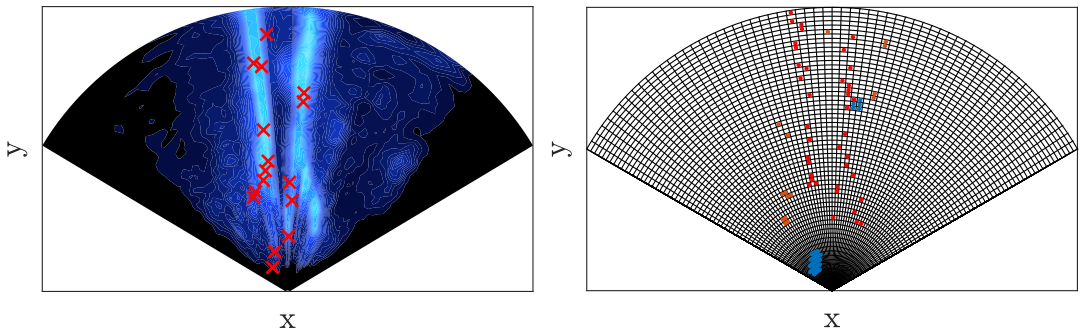
(c) Output, test frame 2.

(d) Sampling, test frame 2.



(e) Output, test frame 3.

(f) Sampling, test frame 3.



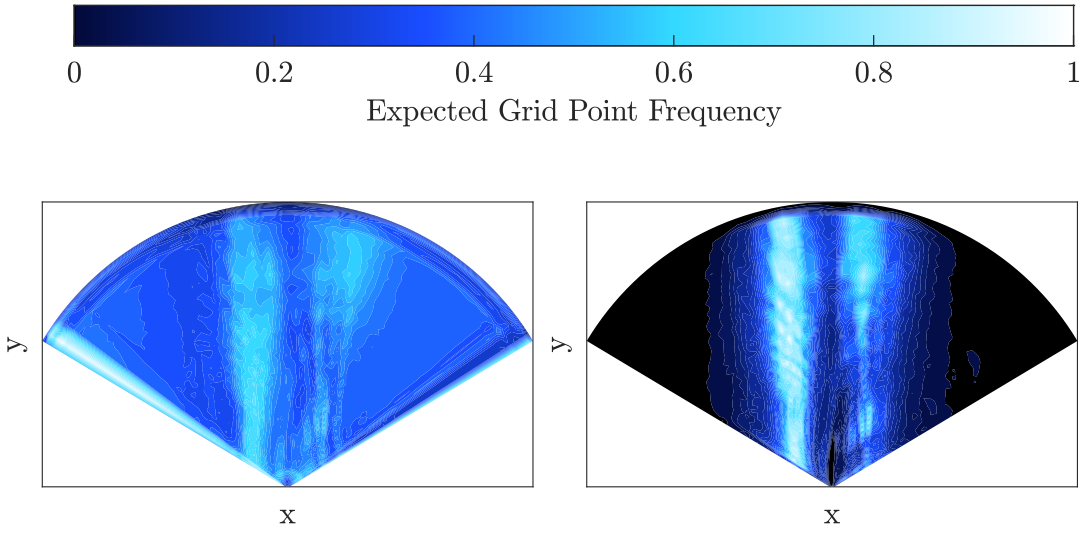
(g) Output, test frame 4.

(h) Sampling, test frame 4.

Figure 4.3: Output from the VAE for test frames 1, 2, 3 and 4. Each row represents one frame, where the VAE output distribution is given as a heatmap in the first column and the subsequent sampling from the distribution is given in the second column. True detections are included as red crosses in the heatmaps in column 1 for comparison. The sampled detections and the conditional input in column 2 are described by the colours as seen in Figure 3.7. The sampling from the heatmap in column 2 was performed using the naive sampling method described in section 3.4 with threshold 0.4. Compare with the photos of the test frames in Figure 3.11 and the conditional input for the frames in 3.12.

4.2 Predictions from the Conditional Generative Adversarial Network

The best performing radar model based on a GAN architecture is described in section 3.3.2. This model generated results for all frames in the test set, as described by Figures 4.4 and 4.5, and instance results for the chosen test frames, as shown in Figure 4.6. The model learns how to map a given conditional input to a possible radar output. It displays a high capability of changing its output depending on the conditional input from the lidar, and does not merely perform an identity mapping.



(a) Output from network. **(b)** After thresholding.

Figure 4.4: Heatmap over predicted detections from the GAN model. Predictions for all frames in the test set have been added to generate the heatmap. In a, the raw output from the network was used. In b, a threshold of 0.8 was subtracted before adding the frames. The result can be compared to the true data in Figure 3.9c. Note that since the output from the model is not binary, but a number between 0 and 1, it is the expected number of detections that is depicted. The GAN model displays some output artifacts at the edges of the grid in a. The most prominent artifact is seen at the left edge, where a clear white line resides.

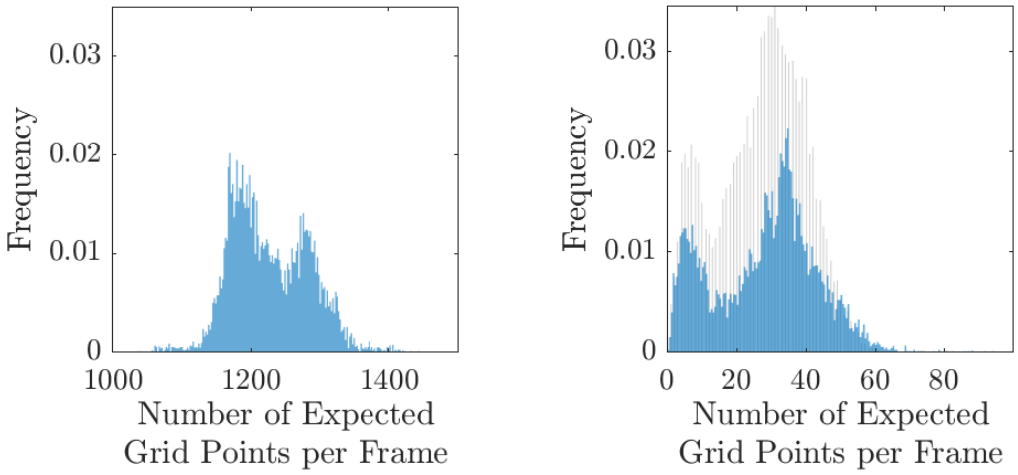
As can be seen in Figures 4.6e and 4.6g, the radar model displays higher probabilities of detection in regions for which there exist real radar detections. Moreover, the regions with lower $P_{\text{detection}}$ seemingly contain less real radar detections. This is most clear in Figure 4.6g, for which few real detections are found in the darker regions, corresponding to a lower $P_{\text{detection}}$.

However, a clear exception from the general correspondence between real radar detections and model $P_{\text{detection}}$ can be seen in Figure 4.6c. Here, real detections are

only generated for the steel barrier to the left of the ego vehicle, which can be seen in Figure 3.11b. The model, on the other hand, displays a high $P_{\text{detection}}$ also for the concrete barrier that the lidar registers to the right and for the barrier noise to the left.

Another exception from the correspondence between the radar sensor and the model is that the model generates a much higher number of expected grid points per frame than the actual number of grid points per frame for the radar. Figure 3.10c displays a frequency for grid points in the range $[0, 60]$ for the radar, while the corresponding Figure 4.5a for the GAN model has a range of $[1000, 1500]$.

To compensate for the high frequency of detections, a threshold is applied before sampling from the generated output, as described in section 3.4. For this model, a threshold of 0.8 is used. When subtracting this threshold before computing the expected number of grid points, the histogram in Figure 4.5b is obtained. Here, the number of expected detections lie in the same range as for the radar data in Figure 3.10c. Moreover, the shape of the distribution in Figure 4.5b for the GAN model is very similar to that of the radar data.



(a) Output from network. **(b)** After thresholding.

Figure 4.5: Histograms over the number of detections per frame, as predicted by the GAN model, using all test frames in the data set. Figure 4.5a depicts the distribution from the raw output from the network, before any sampling has been performed. In Figure 4.5b, a threshold of 0.8 has been subtracted from all probabilities before computing the expected number of detections. The true number of grid points per frame, i.e. corresponding to Figure 3.10c, is included in grey in the background for easier comparison.

In the right column of Figure 4.6 the model output for the four test frames has been sampled with a threshold of 0.8. The lidar data given as input is also displayed in the column to depict the correspondence between sampled output and conditional

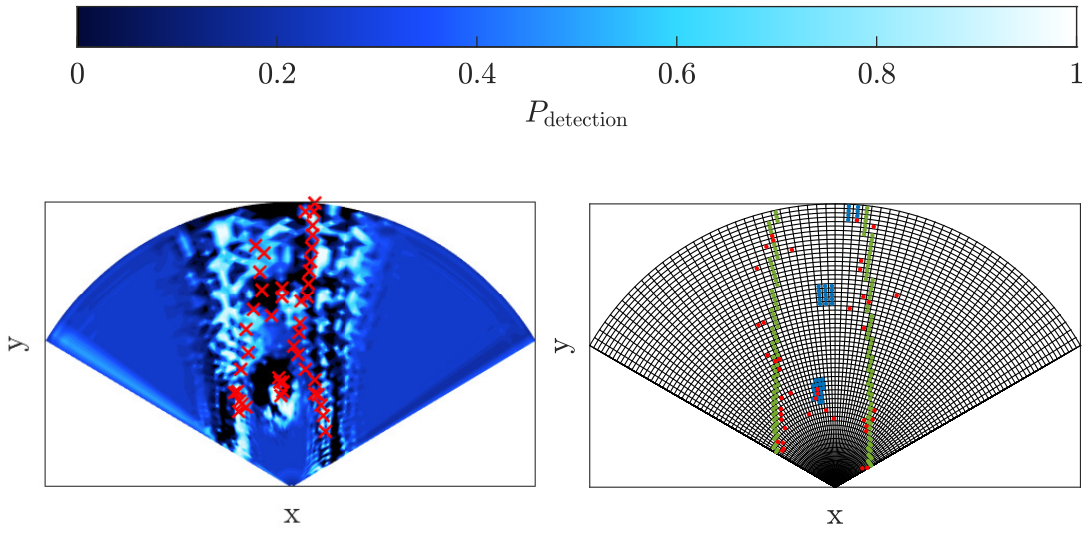
input. The plot colours for the lidar and radar data are explained in Figure 3.7. Visibly, there is a correspondence between the output from the GAN model and the conditional input. For example, this clear correspondence is apparent in Figure 4.6f for which a number of model detections are generated with a clear correspondence to the two closest vehicles and to the barriers seen in the conditional input.

A more curious observation that can be made for the results of the GAN model is that it does not generate a zero probability of detection in the grid margins. This differs from the behaviour of the real radar data, as can be seen by comparing the grid point frequency of the radar sensor in Figure 3.9c with the expected frequency of the radar model in Figure 4.4. Seemingly, the model never generates a $P_{\text{detection}}$ of zero in regions for which there are no lidar grid points. Rather, the model only generates a zero probability of detection in the close presence of a high detection probability. This is most prominent in Figure 4.6h for which there are only two objects. In this figure, clear dark shadows are present behind brighter areas in the point of view of the ego vehicle. In regions where no lidar grid points are present, a constant probability of detection of ~ 0.3 is given.

Moreover, the GAN model displays a set of artifacts that are most prominent in Figure 4.4a, for which several model frame predictions have been added. These artifacts also have a subtle presence for the test frames in Figure 4.6. The artifacts reside on all of the edges of the polar grid, of which the most outstanding one is seen as a white stripe on the left grid edge in Figure 4.4. There is also a darker blue stripe on the right edge of the grid, along with synthetic-looking stripes on the upper edge of the grid. These artifacts make it obvious that the output has not been generated by a real radar sensor. See section 5.3 for a deeper discussion pertaining the GAN artifacts.

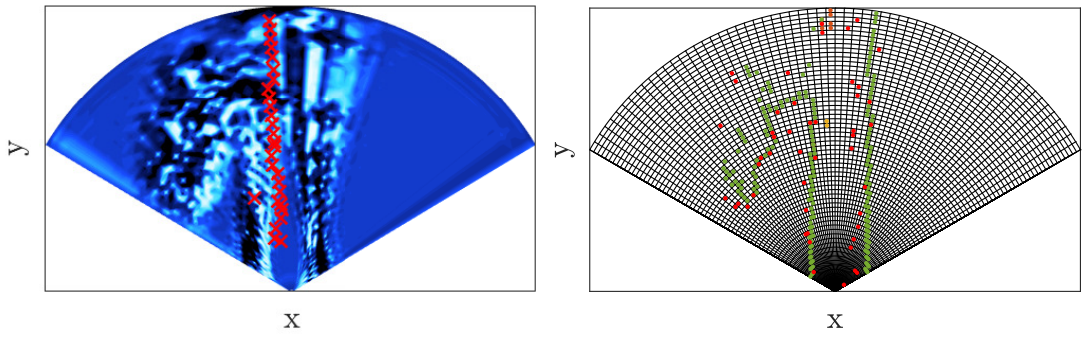
After thresholding with 0.8 for the GAN output, the artifacts disappear and the model heatmap becomes more similar to that of the radar data, as can be seen for the heatmap of the model and the radar data in Figure 4.4b and Figure 3.9c. Moreover, the thresholded heatmap in Figure 4.4b is similar to the heatmap of the lidar in Figure 3.9d. This clearly shows that the GAN model output corresponds to the conditional input.

Lastly, an apparent difference between the results for the GAN model and the radar data is that the model predicts more detections on a short range, close to the host vehicle. As can be seen in Figure 3.9c, the radar data has a grid point frequency that is essentially zero for short ranges. This is not the case for the radar model, which may be observed in both Figures 4.4a and 4.4b. In this sense, the model behaviour is more similar to that of the lidar data, which does not generate a zero grid point frequency for short ranges, as seen in Figure 3.9d.



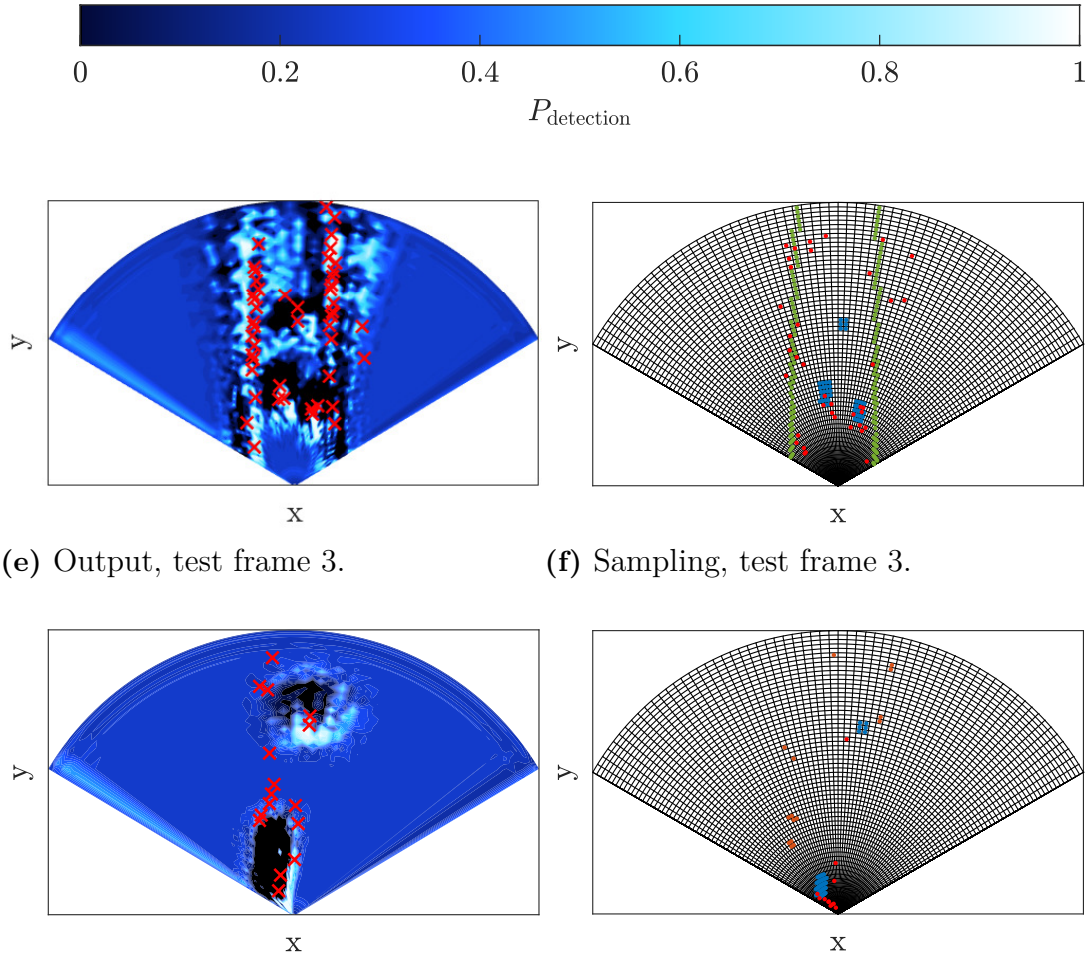
(a) Output, test frame 1.

(b) Sampling, test frame 1.



(c) Output, test frame 2.

(d) Sampling, test frame 2.



(e) Output, test frame 3.

(f) Sampling, test frame 3.

(g) Output, test frame 4.

(h) Sampling, test frame 4.

Figure 4.6: Output from the GAN for test frames 1, 2, 3 and 4. Each row represents one frame, where the GAN output distribution is given as a heatmap in the first column and the subsequent sampling from the distribution is given in the second column together with the conditional input from the lidar. True detections are included as red crosses in the heatmaps in column 1 for comparison. The sampled detections and the conditional input in column 2 are described by the colors as seen in Figure 3.7. The sampling from the heatmap in column 2 was performed using the naive sampling method described in section 3.4 with threshold 0.8.

5

Discussion

With the methods described in Chapter 3, a dataset and neural network models were created and developed. In turn, these generated the results presented in Chapter 4. What remains is to motivate, evaluate and discuss the chosen methods and results. This also leads to propositions of future work that may improve on radar modelling with deep learning methods.

5.1 Sensor Capacities and Data Limitations

Since the radar models were trained on data from radar and lidar, they were limited by the capacities of these sensors. For example, the lidar used for this project has a shorter range than the radar sensor. Hence, the range of the radar model, r_{max}, was chosen with respect to the range of the lidar. Apart from the range, there are several other sensor limitations to take into regard when defining the limitations of the radar models. These include the sensor errors, lidar object classification limitations, azimuth limitations, weather limitations, occlusion limitations, and so on. A thorough understanding of the sensor limits, on which the models are constrained, is necessary to argue for the validity of our radar models. In the sections below, we will dvelve into some of the aspects of this.

5.1.1 Lidar as Conditional Input

Lidar data was used as conditional input to describe the environments of the radar models. This is a serious constraint. Ideally, the conditional input should give an exhaustive and accurate description of the environment of the radar sensor or model, such that the information generated by the radar sensor or model is a subset of the environmental description. That is, no information detected by the radar should be missed by the lidar. Because of this, the radar data was formatted to be limited by the perception field of the lidar, such that it complied with lidar limitations, for example with regard to range.

Apart from imposing sensor limitations on the models, the lidar data is not perfect, and hence it may violate these limitations and/or be erroneous. For example, in Figures 3.11d and 3.12h the lidar completely misses a concrete barrier and appendix B displays a frame for which the lidar detects vehicles that do not exist. In Figure 3.12d, the lidar mistakenly labels the trees on the left side of the road as barriers. Moreover, the sizes of lidar objects are often distorted at a distance. A common case is that the lidar only detects the back of a car, such that the car appears to have the correct width while being far too short lengthwise. See for example appendix B. The radar and lidar may also spot different things within the same sensor limitations. For example, the radar sensor is able to see through objects, while the lidar is not.

Additionally, the lidar information is not exhaustive, such that it for example does not specify whether a barrier is made out of concrete or steel. This information is important to the radar, as can be seen in Figures 3.12c and 3.12d where no radar detections are given for a concrete barrier, while several detections are given for a steel barrier in the same frame. As a result, both the VAE and the GAN radar models perform poorly for this frame, as can be seen in Figures 4.3c and 4.6c. Consequently, material parameters also form a necessary input to the model, as the model cannot figure out whether an arbitrary barrier should be detected or not by itself.

The consequence of using lidar data for environmental descriptions is that the radar models developed with this assumption will try to compensate for the fallacies of the lidar. E.g. the models may ignore some barriers at random instances, or display possibilities for detections in areas for which the lidar generally misses objects. The latter case is very apparent for the GAN model results in section 4.2, as the model never indicates a zero probability of detection in regions that lack lidar data. These model compensations might lead to issues if the model is to be used in a simulation environment, where the conditional input actually is without fallacies.

5.1.2 Finding Accurate Sensor Limitations

As noted in the previous section, it is important to find accurate limitations of the sensors used to generate model data. For example, it could be argued that the range for the data grids should have been shorter than $r_{\max}$, since the used lidar fails to properly detect object dimensions after a range of approximately $\frac{2}{3}r_{\max}$.

Different methods can be used to define accurate sensor limits. For example, the provided sensor capacities may be used as limits, or a limit can be defined based on statistical data for the sensors. For this thesis, the latter approach was used to find that the lidar performed adequately for a maximum range of $r_{\max}$. However, a more thorough investigation of sensor limitations than the one performed for this thesis is possible, and should be beneficial to the radar model performances.

5.1.3 Choice of Data Format

The decision to use the grid data format, as described in section 3.2.2, will be motivated here. Originally, the radar and lidar data was given as lists of detections and objects in every frame. For this thesis, the radar and lidar data was formatted from a list format to a grid format. The motivation for the decision to use the grid data format is based on an analysis of the different possible formats and of the desired neural network input format.

The list format is seemingly more manageable, and fits into the radar tracking algorithm that is to be applied to the radar detections. It also avoids the problem of the approximate positions in the grid. However, to be able to formulate a sensible loss function from such a list, one would have to know which true detection list entry corresponds to which generated detection list entry, and compute the distance between them. Also, if the output is a list, the fact that it is of variable length needs to be accounted for. Moreover, it is difficult to describe the shape of listed objects. While the area of vehicles of reasonable size may be described in a list using their bounding boxes, barriers are harder to describe in a list.

In order to avoid the issues associated with working with list inputs, grid inputs can be used instead. When using grids, no correspondences need to be found between generated and true detections, since one can simply compare the values in each position in a grid to the true ones. The grid format also allows for any number of generated detections and makes it easy to describe object shapes. However, a disadvantage of using a grid format is that a new layer needs to be added to the grid for each added object feature. To describe e.g. position, object class, material information and velocity, four layers are needed. This results in many layers if many features are desired, and might become strenuous to compute. Moreover, the grid format risks creating sparse data, i.e. many elements in a grid may be zero.

Seemingly, the grid format is more suitable for an input to a neural network. Firstly, with grids as input, where the ordering of the grid entries matters, convolutional layers can be applied to guide the network in understanding the spatial association between grid points. Moreover, the connection between the conditional input and the radar output should be easier to understand if the data from both lidar and radar are described in a spatial grid instead of as separate list entries. Also, the grid format made it easy to use residual blocks in the GAN models, and as described in section 2.3.2, residual layers generally make GAN training easier.

The choice of polar coordinates in the grid is motivated by the fact that the requirements on the accuracy of a sensor model in general are stricter in the vicinity of the ego vehicle. Moreover, the shape of the polar grid matches the field of view of the radar sensor. The disadvantage of polar coordinates is that the resolution of the data is low for long ranges in the grid, such that the position for objects far away may be given within a range of several meters. However, this minimum resolution also depends on the limitations of the number of rows and columns in the polar grid.

The data representation was limited to 64 rows and 64 columns in the grids. This leads to some approximations in the positions of the detections and objects and is particularly noticeable in the far end of the grid, where the boxes are larger due to the polar coordinates. More rows and columns would give more detailed data, but also increase the file size and training time.

With the chosen data format, the radar models were able to produce reasonable results, as seen in Chapter 4. Moreover, the format of the plotted data in section 3.2.4 is adequate, such that objects generally scale well over different ranges and e.g. barriers and buildings are sufficiently described. Based on the architecture of obtained models and their respective results, a grid format was optimal for this thesis and seems to be a good choice for future work on radar modelling with neural nets.

Naturally, there is potential to improve upon the chosen data format. For example, it should be beneficial to input a more detailed description of the environment to the model. A 3D grid describing the height of the different objects might help the model generate representative detections. However, this would extend the training time further as the size of the data to be processed is increased.

5.1.4 Reasonable Expectations of the Results

Given the limitations discussed previously, it is clear that a model trained on the data used in this project can not be expected to perform perfectly, since the data itself is not perfect. For example, the lidar could miss an object that the radar detects. When the model is trained on such a frame, it learns that it sometimes can be beneficial to generate detections in places where the lidar has not registered any objects. This property is not desired in the radar model.

Moreover, the networks can not generate a binary output similar to the input, without making any round offs. Hence, we have chosen to interpret the output as a probability, from which detections can be sampled. This is not optimal, as the generated output always will differ in character from the input. Further, the networks will not output a *real* probability distribution, in the sense that they are not trained for probability distributions. Instead, the model output is something between a probability distribution and point-wise detections. Possibly, the network could benefit from receiving an input that is an actual probability distribution, or from receiving an input that has been *softened* by for example adding noise to the binary data.

5.2 Analysis of the Conditional Variational Autoencoder

The results in section section 4.1 show that a CVAE has the potential to learn some of the properties of a real radar sensor. A clear correspondence to the general behaviour of the radar data is visible, such as less detections in the vicinity of the ego vehicle. The model is able to output frames that are similar to this average behaviour. Though, the distribution over the number of grid points per frame does not correspond to the true one, even after thresholding.

A well known problem when using VAEs is that the results tend to be blurry and smudged. This is confirmed by the results in section 4.1. Even though the CVAE is able to learn some useful properties, the generated output is too general and there is not much variance between the frames. The network is strongly bound to the general behaviour of the radar data, and lacks the ability to properly adapt to the conditional input.

The main issue with using CVAEs for this problem is to define a proper loss function. It is desirable to have a generated output that is similar to the real detections, but they do not necessarily have to be exactly the same. For example, a generated detection that is right next to a true detection in the grid might be just as good as if it was generated in the exact same spot. This is hard to define in a loss function.

5.3 Analysis of the Conditional Generative Adversarial Network

As can be seen in section 4.2, the CGAN was able to learn to generate something that is similar to a radar output for a given environment. While the model output is not exactly the same as the radar output, a clear correspondence can be observed between model and sensor, especially after thresholding. However, some disadvantages can be observed with the model, and there is potential for improvement.

A great fallacy of the CGAN model is that it is prone to generate artifacts, as can be seen in Figure 4.4a. It is unclear as to exactly what is causing these artifacts. Possible causes may be boundary conditions for the convolutional layers, the generator architecture, the discriminator architecture, or lack of convergence during training. It is not uncommon to have artifacts for deep convolutional networks, e.g. *checkerboard artifacts* may occur due to transposed convolutional layers, as discussed by the creators of EnhanceNet [9]. However, the artifacts seen in Figure 4.4a are not of the checkerboard type, and may instead originate from some other layer of the CGAN. An investigation should be performed to try to determine the cause of the

artifacts and to remove them, as they clearly should not be a part of a real radar output.

When a threshold is applied to the CGAN model output, these artifacts disappear, as can be seen in Figure 4.4b. Moreover, with thresholding, the model output becomes similar to the radar data both with regard to the heatmap in Figure 4.4b and to the histograms in Figure 4.5b. Consequently, if the CGAN model is to be implemented to model a radar, it should be used together with some thresholding method.

Another potential fallacy of the CGAN model is that it has learnt to compensate for the shorter range of the lidar. That is, the model never generates a zero probability of detection in areas where lidar data is lacking. This clearly points to the flaws in using the lidar data for environmental descriptions, as the CGAN model has learnt that objects still may exist in regions for which the lidar does not register any objects. This fact is partly compensated for when subtracting the threshold, as can be seen in Figure 4.4b. Here, the expected grid point frequency is zero at the edges of the grid.

Moreover, it is clear that the CGAN model performance could be improved by including material parameters of registered lidar objects. As can be seen in for example Figure 4.6c, the model predicts detections for a concrete barrier, even though a real radar sensor would not detect it. This is because the model treats all barriers the same, without knowing whether they are made of concrete or steel.

Lastly, the CGAN model fails to capture some of the general properties of the radar data. For example, as can be seen in Figure 4.4, the network has not learnt to omit detections within short range. This may be a result of the fact that the network was trained on too few epochs. Compared with the VAE which was trained for 30 epochs, the CGAN was only trained on 5,120,000 frames, corresponding to approximately 4 epochs of the dataset.

In conclusion, while the CGAN displays some errors and has potential for improvement, it is evident that this network has the capacity to model a radar. The model shows a clear correspondence to conditional input given by the lidar, and it has a behaviour that is similar to that of a radar sensor. Also, a clear advantage of using the CGAN model is that its generating part does not need to be trained for an explicit loss function. This enables the model to learn the actual radar behaviour, without any explicit limitations.

5.4 Comparison of Model Results

Overall, the VAEs and GANs used in this project have displayed the types of problems and properties that are well known for these types of generative networks. For

example, the output from the VAE was blurry, but captured the average behaviour of the data. On the other hand, the GAN was better at producing samples with fine resolution, but had a complex training process and did not necessarily converge.

Based on the results presented in Chapter 4, the GAN model seemingly fits the purpose of the radar model better. Its main advantage is that it shows more correlation to the conditional input. Also, when using GANs, the problem of defining a loss function is circumvented. The distribution of expected grid points per frame generated by the GAN, in Figure 4.5, also has a clearer correspondance with the real radar detections than for the VAE in Figure 4.2. However, the GAN model lacks some of the general radar properties that the VAE displays, such as the lack of detections near the ego vehicle and a low probability of detections at the edges. Nevertheless, the latter issue seems to be avoidable by applying sampling, as can be seen in Figure 4.4b.

Given the results of this thesis, a network that combines a GAN and a VAE seemingly has the potential to perform even better at producing a satisfying output. Moreover, different types of coalitions between VAEs and GANs have already been used in previous research [5, 27]. A common approach is to replace the generator in a GAN with a VAE. However, this is not trivial to realise. Even though the networks proposed in this project produce somewhat similar outputs, their training procedures are very different. Since their respective objective is different, the VAE loss and the adversarial loss might end up in a situation where they pull the network in different directions.

5.5 Performance of the Evaluation Methods

The question remains as to what the correct behaviour of a radar model should be. Two different methods to define this for a network, either with a loss function or by the employment of a discriminator, were used. For the analysis of the performance of a radar model, a combination of the overall statistics of the model output was studied alongside direct outputs for some test frames. Primarily, the performance of the models was evaluated using the perceptual senses of the authors. Clearly, more work on defining how a radar sensor should behave for different scenarios will benefit the development of accurate radar models.

Additionally, the radar models should be evaluated with regard to the objective of the thesis, i.e. to create a model of a radar for facilitation of virtual verification of ADAS and AD systems. The optimal approach to this would be to study the existing radar models used for virtual verification, as self-driving systems already are tested virtually, and to compare them to the radar models created for this thesis. Presumably, there should already exist initial knowledge pertaining the validation methods for virtual verification systems, and it would be interesting to apply it to the performance of the developed radar models. However, with the methods

used in this thesis, the performance of the radar models has only been evaluated with regard to that of real radar sensors. Thus, further research with respect to the validity of virtual verification with sensor models is necessary before the radar model performance can be put into perspective with the objective of this thesis.

5.6 Future Work

Although some promising results for modelling a radar using VAEs and GANs were obtained in this project, additional work is needed before the models can be used for virtual verification. Some of the potential aspects for accuracy improvement have already been discussed in the previous sections. These concern more accurate data, more information included in the data, further network development, more training time and more accurate evaluation methods. Additional examples of future work on radar modelling that are related to the enablement of virtual verification are described below.

First of all, the output from the models in the thesis need to be converted to a binary format. A better method for sampling from the output is needed. Unlike the naive sampling method used in this project, where detections where sampled independently of each other, a rigorous sampling method should take detections in the vicinity of a possible sample into account.

Also, to be able to apply a standard tracking algorithm on detections, velocities are needed. To model this, the velocity of each detection could either be given as an additional input and output from the network, or the model could be adjusted to work with series of frames.

6

Conclusion

In this thesis, deep generative networks were used to model a radar sensor in automotive scenarios. Two different network structures were proposed, a conditional variational autoencoder and a conditional generative adversarial network. The conditional variational autoencoder was able to learn where radar detections are more likely, but produced blurred samples and had trouble with generating varying outputs for different frames. The conditional generative adversarial network gave more fine grained radar samples that corresponded to the conditional input, while it did not completely converge and lacked some of the average radar behaviour that the VAE could display. We can conclude that neural networks have the potential to model radar sensors for the set up described in this project, and that the generative adversarial network holds the most potential of the tried networks. However, before the models can be of use for virtual verification, more work needs to be done in creating a rigorous sampling method for the network output and in deriving more accurate evaluation methods.

Bibliography

[1] Volvo Car Group. *Pilot Assist and Lane assistance*. 2019. URL: https://www.volvocars.com/uk/support/article/3395a91b40bddd9ac0a801511916dab3 (visited on 04/25/2019).

[2] Volvo Car Group. *Park Assist Pilot*. 2019. URL: https://www.volvocars.com/uk/support/article/354d21c7f91ab069c0a801511344556b (visited on 04/25/2019).

[3] Maria Caspani and Paul Lienert. "Americans still don't trust self-driving cars, Reuters/Ipsos poll finds". In: *Reuters* (Apr. 1, 2019). URL: https://www.reuters.com/article/us-autos-selfdriving-poll/americans-still-dont-trust-self-driving-cars-reuters-ipsos-poll-finds-idUSKCN1RD2QS (visited on 04/25/2019).

[4] Nidhi Kalra and Susan M. Paddock. "Driving to Safety - How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability?" In: *RAND Corporation* (2016).

[5] Tim A Wheeler et al. "Deep stochastic radar models". In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2017, pp. 47–53.

[6] Philipp Berthold et al. "An Abstracted Radar Measurement Model for Extended Object Tracking". In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2018, pp. 3866–3872.

[7] Waqas Malik and Alexander Suhre. "Simulating object lists using neural networks in automotive radar". In: *2018 19th International Conference on Thermal, Mechanical and Multi-Physics Simulation and Experiments in Microelectronics and Microsystems (EuroSimE)*. Apr. 2018, pp. 1–5. DOI: 10.1109/EuroSimE.2018.8369885.

[8] M. Buhren and Bin Yang. "Simulation of Automotive Radar Target Lists using a Novel Approach of Object Representation". In: *2006 IEEE Intelligent Vehicles Symposium*. June 2006, pp. 314–319. DOI: 10.1109/IVS.2006.1689647.

[9] Mehdi SM Sajjadi, Bernhard Scholkopf, and Michael Hirsch. "Enhancenet: Single image super-resolution through automated texture synthesis". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 4491–4500.

[10] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[11] Jeremy Hsu. *A New Way to Find Bugs in Self-Driving AI Could Save Lives*. Nov. 3, 2017. URL: https://spectrum.ieee.org/tech-talk/robotics/artificial-intelligence/better-bug-hunts-in-selfdriving-car-ai-could-save-lives (visited on 04/22/2019).

[12] Szu-Yu Chou, Li-Chia Yang, and Yi-Hsuan Yang. "MidiNet: A Convolutional Generative Adversarial Network for Symbolic-domain Music Generation using 1D and 2D Conditions". In: *CoRR* abs/1703.10847 (2017). arXiv: 1703.10847.

[13] Anil K Jain, Jianchang Mao, and KM Mohiuddin. "Artificial neural networks: A tutorial". In: *Computer* 3 (1996), pp. 31–44.

[14] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep Learning". In: *Nature* 521 (May 2015), pp. 436–44. DOI: 10.1038/nature14539.

[15] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *arXiv e-prints* (Sept. 2014). arXiv: 1409.1556.

[16] Christian Szegedy et al. "Going Deeper with Convolutions". In: *Computer Vision and Pattern Recognition (CVPR)*. 2015. URL: http://arxiv.org/abs/1409.4842.

[17] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 249–256.

[18] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[19] Yoon Kim. "Convolutional Neural Networks for Sentence Classification". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Oct. 2014, pp. 1746–1751. DOI: 10.3115/v1/D14-1181.

[20] Yann LeCun and Yoshua Bengio. "Convolutional networks for images, speech, and time series". In: *The handbook of brain theory and neural networks* 3361.10 (1995), p. 1995.

[21] Matthew D Zeiler et al. "Deconvolutional networks". In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2010*. 2010.

[22] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[23] Aaron Courville, Yoshua Bengio, and Ian Goodfellow. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[24] Prakash Pandey. *Deep Generative Models*. 2018. URL: https://towardsdatascience.com/deep-generative-models-25ab2821afd3 (visited on 01/23/2019).

[25] Kevin. *Variational Autoencoders Explained*. 2016. URL: http://kvfrans.com/variational-autoencoders-explained/ (visited on 01/23/2019).

[26] Carl Doersch. "Tutorial on variational autoencoders". In: *arXiv preprint arXiv:1606.05908* (2016).

[27] Jianmin Bao et al. "CVAE-GAN: fine-grained image generation through asymmetric training". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 2745–2754.

[28] Ahmed M. Elgammal et al. “CAN: Creative Adversarial Networks, Generating "Art" by Learning About Styles and Deviating from Style Norms”. In: *CoRR* abs/1706.07068 (2017). arXiv: 1706.07068.

[29] Tero Karras, Samuli Laine, and Timo Aila. “A Style-Based Generator Architecture for Generative Adversarial Networks”. In: *CoRR* abs/1812.04948 (2018). arXiv: 1812.04948. URL: http://arxiv.org/abs/1812.04948.

[30] Grigory Antipov, Moez Baccouche, and Jean-Luc Dugelay. “Face Aging With Conditional Generative Adversarial Networks”. In: *CoRR* abs/1702.01983 (2017). arXiv: 1702.01983. URL: http://arxiv.org/abs/1702.01983.

[31] Martin Arjovsky and Léon Bottou. “Towards Principled Methods for Training Generative Adversarial Networks”. In: *arXiv e-prints* (2017). arXiv: 1701.04862 [stat.ML].

[32] Alec Radford, Luke Metz, and Soumith Chintala. “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”. In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. 2016. URL: http://arxiv.org/abs/1511.06434.

[33] Mehdi Mirza and Simon Osindero. “Conditional generative adversarial nets”. In: *arXiv preprint arXiv:1411.1784* (2014).

[34] Merrill I. Skolnik. *Radar*. 2018. URL: https://www.britannica.com/technology/radar (visited on 01/28/2019).

[35] Merrill I. Skolnik. *Radar Handbook, Third Edition*. The McGraw-Hill Companies, 2008. ISBN: 9780071485470.

[36] Karl Granström and Marcus Baum. “Extended Object Tracking: Introduction, Overview and Applications”. In: *CoRR* abs/1604.00970 (2016). URL: http://arxiv.org/abs/1604.00970.

[37] J. D. Echard. “Estimation of radar detection and false alarm probability”. In: *IEEE Transactions on Aerospace and Electronic Systems* 27.2 (Mar. 1991), pp. 255–260. ISSN: 0018-9251. DOI: 10.1109/7.78300.

[38] R. H. Rasshofer and K. Gresser. “Automotive Radar and Lidar Systems for Next Generation Driver Assistance Functions”. In: *Advances in Radio Science* 3 (2005), pp. 205–209. DOI: 10.5194/ars-3-205-2005. URL: https://www.adv-radio-sci.net/3/205/2005/.

[39] A. Sole et al. “Solid or not solid: vision for radar target validation”. In: *IEEE Intelligent Vehicles Symposium, 2004*. June 2004, pp. 819–824. DOI: 10.1109/IVS.2004.1336490.

[40] K. J. Vinoy and R. M. Jha. “Trends in radar absorbing materials technology”. In: *Sadhana* 20.5 (Oct. 1995), pp. 815–850. ISSN: 0973-7677. DOI: 10.1007/BF02744411. URL: https://doi.org/10.1007/BF02744411.

[41] Veoneer. *Radar*. 2019. URL: https://www.veoneer.com/en/radar (visited on 01/28/2019).

[42] Velodyne LiDAR. *HDL-64E*. 2019. URL: https://velodynelidar.com/hdl-64e.html (visited on 01/28/2019).

[43] Shoujin Wang et al. “Training deep neural networks on imbalanced data sets”. In: *2016 international joint conference on neural networks (IJCNN)*. IEEE. 2016, pp. 4368–4374.

[44] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *arXiv e-prints*, arXiv:1412.6980 (Dec. 2014), arXiv:1412.6980. arXiv: 1412.6980 [cs.LG].

A

Appendix 1 - Test Frames

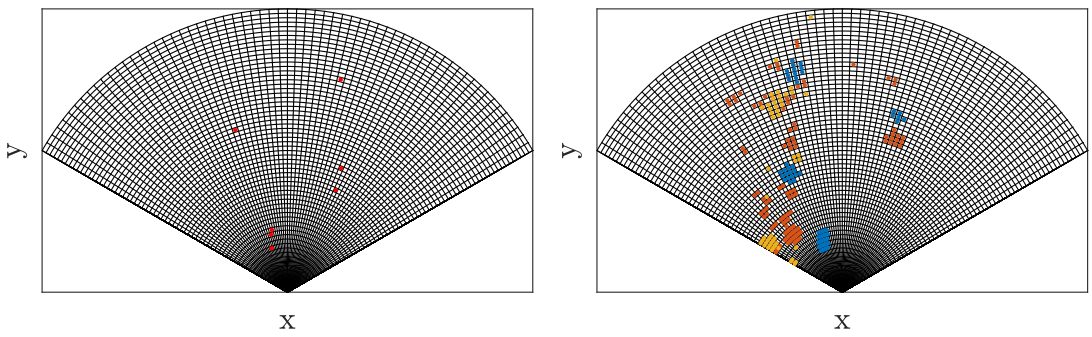
Figure A.1: The ten test frames used for model evaluation. These are sampled from the test set described in section 3.2.3.

B

Appendix 2 - Lidar Errors

For the frame shown in Figure B.1, the lidar erroneously detects non-existent vehicles, as can be seen in Figure B.2. And for the frame shown in Figure B.3, the lidar gives incorrect dimensions of detected objects, as can be seen in Figure B.4.

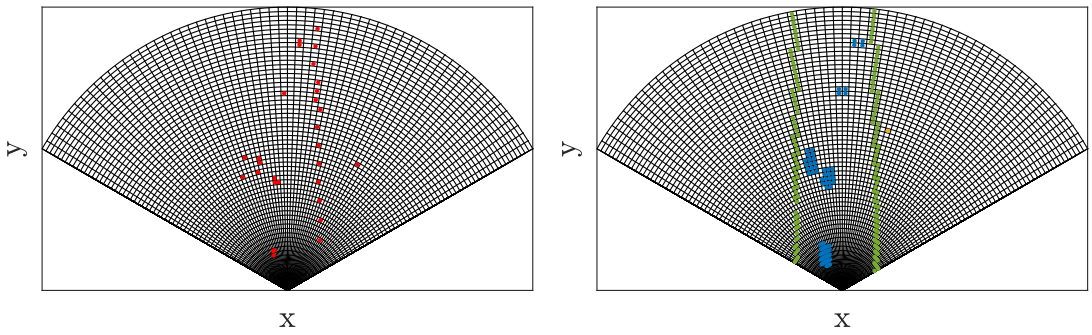
Figure B.1: Photo of a test frame taken from the ego vehicle. There are two other vehicles in the vicinity of the ego vehicle, and no barriers.



(a) Radar Detections **(b)** Lidar Objects

Figure B.2: The radar and lidar grid for the frame in Figure B.1. In b, observe that the lidar registers three non-existent vehicles, in blue, on both sides of the road.

Figure B.3: Photo of a test frame taken from the ego vehicle. There are five other vehicles in the vicinity of the ego vehicle, and a steel barrier can be observed on the right side.



(a) Radar Detections **(b)** Lidar Objects

Figure B.4: The radar and lidar grid for test frame 9. See Figure B.3 for a photo of the scenario. In b, observe that the lidar registers different object dimensions for the two vehicles further away and the three vehicles that are closer to the ego vehicle. The two far-away vehicles are given much smaller dimensions, while they seemingly should not be smaller, as can be seen in Figure B.3.