Evaluation of Deep Neural Networks for Radar Based Point Cloud Classification

Emil Ronkainen
emil.ronkainen@gmail.com
Alexander Sandelius
sandelius.alexander@gmail.com

Department of Electrical and Information Technology
Lund University

Supervisor: Anna Svensson, Johan Sjögren

Academic Supervisor: Johan Thunberg

Examiner: Michael Lentmaier

June 13, 2024

Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

Thanks to the ever-increasing computational power, machine learning has become a staple in computer science. Many computer vision methods have become so reliable that their implementation in the surveillance industry is now the de facto standard for many object detection and classification tasks. The use of machine learning for surveillance is not confined to images and videos, however, it is also applicable to RADAR. Axis Communications produces several RADAR-based solutions, some use RADAR only, whereas others provide RADAR and video fusion technology. RADAR generates unordered sets of points, or point clouds. Point cloud classification is a fairly unexplored area of machine learning, in particular in the context of RADAR generated point clouds. In this thesis, we address the classification of moving clusters of RADAR point cloud data, which requires the classifier to consider several instances of an input cluster, where spatial as well as temporal information is present. We examine how two main classifier architectures perform on this data type and compare them to the existing classifier at Axis. Empirically, they show promising performance on the available data. However, the results also indicate that a more robust study might be required before employing the classifiers.

Popular Science Summary

This study delves into a cutting-edge area: differentiating between humans and vehicles by combining radar devices with machine learning. Our research shows promising results but also highlights the need for more extensive testing to ensure reliability in real-world applications.

Radar surveillance offers the advantage of anonymity by allowing efficient monitoring without capturing personal details, ensuring privacy while still maintaining security. By integrating machine learning, radar systems become even more efficient and intelligent, capable of distinguishing between different objects and predicting potential threats with high accuracy. This advanced technology not only enhances safety and security but also respects individual privacy, providing a seamless and non-intrusive way to monitor and protect our world.

This thesis looks at how to distinguish objects over time using radar. Radar systems create groups of points, or "point clouds", which represent objects in their surroundings. Determining whether these point clouds are vehicles or humans using machine learning is still a relatively new field of research, especially when the data comes from radar. As these point clouds are followed over time, there is even more information than that provided by the radar to consider when tackling this task.

The study tests two main types of machine learning architectures on this data, both based on a technique known as PointNet. These are compared to another technique, developed by Axis Communications.

Our results show that these new techniques perform well on the available data, and provide novel insights into the possibilities of combining machine learning and radar. However, it also shows the shortcomings of these newer techniques, proposing further research opportunities that might find even better outcomes.

Acknowledgements

We would like to extend our most sincere thanks to our supervisors at Axis, Anna Svensson and Johan Sjögren, for their mentorship and sustained support. We greatly appreciate their insights, debugging skills, and great humor, both in the office and over Teams.

We would also like to sincerely thank our LTH supervisor Johan Thunberg, for his unmatched curiosity. His many questions, ideas, and comments helped tremendously and forced us to think deeper about the vital components of this thesis.

A great thank you to VOI electric scooters which helped traverse the mountainous terrains of Lund, saving us the hours needed to complete this thesis in time.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

Thanks to the ever-increasing computational power, machine learning has become a staple in computer science due to its broad spectrum of applicability. The surveillance industry has much to gain from different machine learning methods as it significantly reduces the need for human attention and reduces the margin of error inherent to human interaction. Computer vision methods have become so reliable that their camera-based implementations are now the de facto standard for many image recognition and object detection tasks.

The use of machine learning in surveillance is not confined to images and video but is also applicable to point-based data, such as LiDAR (light detection and ranging) and RADAR (radio detection and ranging, subsequently referred to as "radar"). This master´s thesis will focus on radar. While image recognition uses images with pixels and RGB values, radar uses unordered sets of points (henceforth referred to as *point clouds*), typically in a two- or three-dimensional space. Radar devices provide several advantages over video in the field of surveillance. They are often less sensitive to difficult weather conditions, perform well in poorly lit environments, and allow for greater anonymity [32]. These properties make the radar technique a promising option for object detection within the context of surveillance.

Axis Communications AB (henceforth referred to as Axis), produces several different radar devices. Some are based on radar only whereas others provide radar video fusion technology. These devices are equipped to run object classification and currently employ a classification model that separates human and vehicle objects, developed at Axis. Using data previously gathered by the existing frameworks for data acquisition and pre-processing present at Axis, this work focuses on the last part of the classification pipeline, the classifier itself. We aim to provide an overview of a couple of such classifiers, prevalent in other fields of machine learning for classification. These will be compared with the current model, to determine whether these classifiers prove useful for radar classification.

1.1 Previous work

Here we provide an overview of the radar point cloud classification field by highlighting some recent publications in this area. Some general point cloud classifiers

and their applicability to radar are also discussed here. Lastly, the previous work done at Axis is supplied, seeing as this acts as the main precursor for this thesis.

The field of point cloud classification has seen a great surge of interest in the recent years. Articles such as [22], [30], and [33] all propose methods of using point clouds for object classification. However, like much of the work on point cloud classifiers, these articles consider data gathered by LiDAR rather than radar. While both LiDAR and radar generate point clouds, the output data differs in some important aspects. LiDAR generally provides more spatial information thanks to higher resolution and more sampled points. However, radar includes other measurements such as radial velocity and signal strength. Due to these differences, some properties of the proposed methods may not apply to radar point clouds. Moreover, there is no guarantee that the performance of these models is transferable to the radar field.

One approach not explicitly concerned with LiDAR is PointNet [24]. Following the publication *Order Matters: Sequence to Sequence for Sets* [28] on deep learning for unordered sets, PointNet introduces a novel way of directly using point clouds as input. PointNet studies the inherent features of a point cloud, rather than exploiting task-specific properties of the data, and is thus much better suited as a starting point for this thesis.

As stated previously, radar-specific point cloud classification is a far less explored topic than LiDAR. While much of the literature that is available on the subject, such as [34] and [11], is not concerned with the field of surveillance, it still speaks to the legitimacy of radar-based point cloud classification. [2] utilizes a similar feature vector to us, in combination with the three-dimensional aspect of point clouds, but does not take advantage of the temporal aspect.

1.2 Problem Formulation

While there exist classifiers that have been proven to be proficient for point cloud classification, these are commonly developed for LiDAR data. This thesis concerns point clouds originating from radar devices. More specifically, moving radar point clouds, sampled over time. We aim to investigate whether existing classifiers can be modified to accommodate the change from LiDAR to radar data. Capturing information over time is paramount due to the temporal aspect of the data.

To address this, we explore two main classifier architectures based on the work of [24], further discussed in Section 1.1, employed on a two-class classification problem. We use the two classes "human" and "vehicle", seeing as Axis provides a robust dataset of moving radar point clouds for these classifications.

The limited amount of previous work on moving radar point clouds warrants employing multiple variants of each architecture to provide a more robust insight into how different classifiers interact with the data. To address this, we propose as the overarching problem or objective to examine six different classifiers, comparing these to the current Axis classifier, which acts as a baseline for the evaluation.

We intend to evaluate and compare the applicability of each potential classifier by analyzing the following aspects

1. Classification accuracy

2. Reliability
3. Resource demand

Classification accuracy refers to the classifiers' capacity to successfully classify point cloud inputs, while reliability quantifies the robustness of this accuracy. The resource demand metric considers the computational requirements of a classifier.

1.3 Work Outline

Chapters 2 and 3 provide the necessary preliminaries to understand the descriptions, results, and discussions in subsequent chapters. Chapter 2 focuses on the machine learning aspect of the work, while Chapter 3 features a brief explanation of radar technique.

Chapters 4 and 5 outline the data and properties of the models in the thesis, respectively. Chapter 4 summarizes the pipeline of pre-processing steps applied to the raw radar data and the prevalent characteristics of the resulting data. Chapter 5 describes the architecture of the classifiers examined in this work.

Chapter 6 details the approach for model training and hyperparameter optimization. The latter part of the chapter presents and briefly explains the results of the experiments.

Finally, Chapter 7 discusses the implications of the results in Chapter 6 and motivates design decisions made during the experiments. A section outlining continuations on this thesis and other future work is also provided here.

Chapter 2
Machine Learning

2.1 Fundamentals

Here, some of the essentials within the area of machine learning, specifically the tools and methods pertinent to the later introduced models (see Chapter 5), are introduced to provide the required context for the remainder of the thesis. Much of the material in this Chapter is based on [12] and the interested reader may consult that book for greater detail on the topic.

2.1.1 Machine Learning Model

A machine learning model aims to automate, or learn, the process of discovering important relationships and patterns between features of a problem space [15]. It does so by iteratively applying its algorithm to problem-specific data and evaluating its performance. This contrasts early AI which relied heavily on explicit sets of rules defined by humans, which defined the behaviour of the computer [12].

When building a handcrafted analytical model, i.e. a quantitative model designed to accomplish a specific task [3], the process can be split into two parts: **feature extraction** and **model building**. Feature extraction seeks to pinpoint important aspects of the input samples to provide the model with a suitable representation of the problem, whereas model building refers to the act of converting the previously defined features into outputs. In machine learning, the model designer provides a set of trainable parameters that the model *learns* by evaluating itself on data from the problem space. A machine learning model that learns only the model building step, and thus still relies on manual feature extraction, is often referred to as a *shallow* machine learning model. Conversely, a model that learns both feature extraction and model building is known as a *deep* machine learning model. Note that while feature extraction is automated by deep models, respecting external knowledge of features in the input can still prove useful for performance. [15]

2.1.2 Data and Error

To measure model performance, it is common to introduce an **error**, or **loss**, function. The error function quantifies how much the model output differs from

the expected output [14]. During training, a model can be tasked with tuning its trainable parameters such that they try to minimize the error function in order to increase performance. One important loss function in the context of classification is the *Categorical Cross-Entropy*. For a model with K outputs and trainable parameter vector $\boldsymbol{p}$, the categorical cross-entropy error of N samples is given by

$$CE = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \ln y_{nk}, \quad \text{where } y_{nk} = f_k(\boldsymbol{p}) \tag{2.1}$$

where t_{nk} and y_{nk} is the expected and predicted value of output k for sample n, respectively, and f_k is the output function of output k [5].

Model input is commonly split into two separate sets of data: **training data** and **test data**. The model parameters $\boldsymbol{p}$ are tuned solely on the training data while the test set is reserved for evaluation. To draw any useful conclusions about how the test set performance is affected when only the training data is observed, some assumptions about the data acquisition process have to be made. Common practice is to assume that both sets of data are collected under the **Independent and Identically Distributed** (i.i.d.) assumptions [12]. This means that samples in the data sets are *independent* from each other and that the sets are *identically distributed*, drawing from some shared, underlying distribution. Hence, under the i.i.d. assumptions, the expected error for the training and test data will be equal, since both expectations are derived from the same data acquisition process. In practice, however, the test set is not evaluated on until the model parameters have been learned on the training data. Thus, the parameters of the model are dependent on the training data, and the expected error for the test data will not necessarily be equal to the training data. The learning process is often referred to as *training* the model.[12]

Datasets can be constructed in several ways, depending on the learning paradigm. There are two main paradigms, *supervised* and *unsupervised* learning. Supervised learning requires that, for a given input, the corresponding expected output, or label [12], is also provided by the dataset [15]. Conversely, unsupervised learning assumes that this label information is not available, or ignored at the time of training [8].

2.1.3 Capacity

Capacity is an informal way of defining to which extent a model can represent different underlying distributions of data [12]. A model with low capacity is very limited in its ability to represent different distributions whereas high capacity models might fit properties of the training data which are not present in the test set. Insufficient capacity, more commonly referred to as *underfitting*, and abundant capacity, or *overfitting*, can be identified by observing the error on the training and test sets. Underfitting will yield high errors for both datasets due to it not being able to capture essential feature trends [8]. Overfitting is distinguished by a low training error but a high test error. This indicates that the model learns not only the important features of the training set but also random noise in the training data. An example of how a function is underfitted and overfitted to a set of points is shown in figure 2.1.

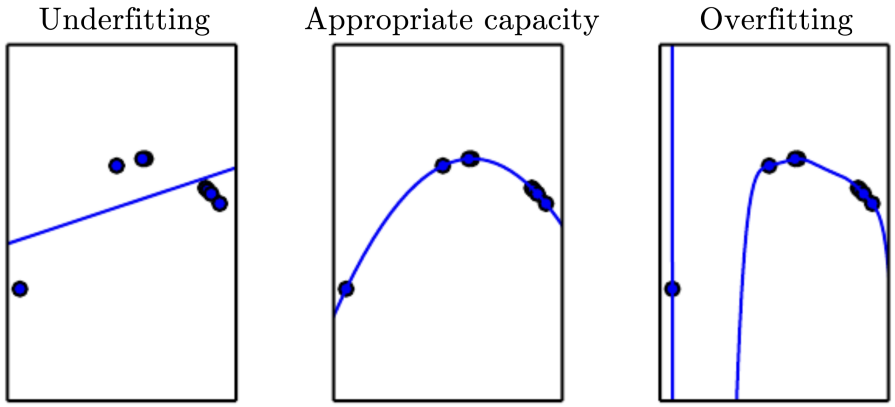


Figure 2.1: A visual example of how a second-order function is interpolated using too low (left), too high (right), and appropriate (middle) capacity (Image from [12], Chapter 5.2).

Ultimately, one wants the model to perform well on previously unseen input [15]. This *generalization performance* [12] can be determined by evaluating metrics such as error on the test data, seeing as this set has not been involved in the learning process.

2.1.4 Hyperparameters

Hyperparameters are, unlike regular parameters, not tuned by the model during training. Instead, hyperparameters are set beforehand, either by some external framework or manually by a model designer [23]. Sometimes, these parameters are inherently hard to optimize, such as the choice of error or objective function. More frequently, however, a hyperparameter is related to the capacity of the model [12]. Tuning capacity parameters during training might provide great results initially, seeing as the model can increase its capacity whenever it deems the underlying distribution of data too complex to fit with its current capacity. However, as mentioned in Section 2.1.3, a too large capacity might incentivize overfitting. Thus, excluding the tuning of these parameters from the learning process reduces this risk significantly.

Tuning hyperparameters poses an interesting task. Using the training data for tuning is, as previously discussed, infeasible. However, the test data is not an option either, seeing as it would no longer retain its property of being previously unseen data. Common practice is to introduce a third set of data, the *validation set*, tasked with this problem specifically [12]. The validation set is typically taken as a subset of the training data.

2.1.5 Evaluation

This section introduces important metrics and procedures for evaluation. In the context of this work, the two-class classification models considered provide two outputs: the predicted class and the prediction probability, or confidence, of the predicted class.

The confusion matrix is one of the most prevalent procedures of presenting classification results [19]. For a two-class problem, the confusion matrix can be defined as

$$\begin{bmatrix} m_{AA} & m_{AB} \\ m_{BA} & m_{BB} \end{bmatrix} \tag{2.2}$$

where, for any i and j, m_{ij} denotes the number of samples from class i classified as class j. Hence, when $i = j$, the corresponding value represents the number of correctly classified samples of class i whereas when $i \neq j$, the value indicates the number of class i samples incorrectly classified as class j. Alternatively, m_{ij} can be given as the fraction of the total amount of samples from class i, classified as class j.

Many useful metrics can be derived from the confusion matrix. One such metric, perhaps the most fundamental for classification problems, is the *accuracy* (ACC). Accuracy is calculated as the fraction of samples along the main diagonal, i.e. the fraction of correctly classified samples. Using the confusion matrix, this is defined as

$$ACC = \frac{m_{AA} + m_{BB}}{m_{AA} + m_{AB} + m_{BA} + m_{BB}}. \tag{2.3}$$

Two other well-established measurements obtained from the confusion matrix are *precision* (PRC) and *recall* (RCL) [19]. For some class A, precision is the fraction of class A predictions that are correct and recall the fraction of correctly identified class A samples [12]. These are derived from the confusion matrix as follows

$$PRC = \frac{m_{AA}}{m_{AA} + m_{BA}} \tag{2.4}$$

$$RCL = \frac{m_{AA}}{m_{AA} + m_{AB}} \tag{2.5}$$

While accuracy provides a very intuitive way of interpreting model performance, metrics such as precision and recall help evaluate the robustness of the accuracy [12]. In cases where there is an imbalance of samples between classes, a model might favor only guessing the majority class, seeing as this achieves good accuracy. However, this would result in a recall score of 0 for the minority class, suggesting that the model fails to learn the properties of this class.

In addition to the above-mentioned metrics, one can consider the prediction probability a measure of robustness for a classifier. This is subsequently referred to as the confidence of a model.

2.2 Artifical Neural Networks (ANNs)

A prevalent subset of machine learning models is **Artificial Neural Networks** (ANNs), where the models provide mappings from input data to one or several output variables through a composition of functions [7]. The function composition is dependent on a parameter vector $\boldsymbol{p}$, which is learned by the network [12]. Each part of the function composition produces an intermediate representation, or embedding, of the input data by sequential application of a linear transformation and a non-linear output (activation) function [5]. The linear transformation and output function combination is known as a *layer* [12] and there exist various types of layers, each providing advantages for different types of input data. It is common to represent ANNs as Directed Acyclic Graphs (DAGs) using vertices for inputs and outputs while weights are conveyed by edges.

In this thesis, the evaluated models are *deep* neural networks. As eluded to in section 2.1.1, deep models learn, in contrast to shallow models, to extract their own set of features from the input data before calculating an output. The notion of deep networks stems from the multiple function compositions not present in shallow ANNs [7].

2.2.1 Activation Function

The activation functions introduce non-linearities into the network [5]. This non-linearity captures properties in the data that would otherwise be difficult or impossible to extract [12]. Without activation functions, the composition could be expressed as a single, linear layer

$$ABCDx = Ex \quad \text{where } E = ABCD \tag{2.6}$$

where $A, \ldots, E$ are linear transformations and x input.

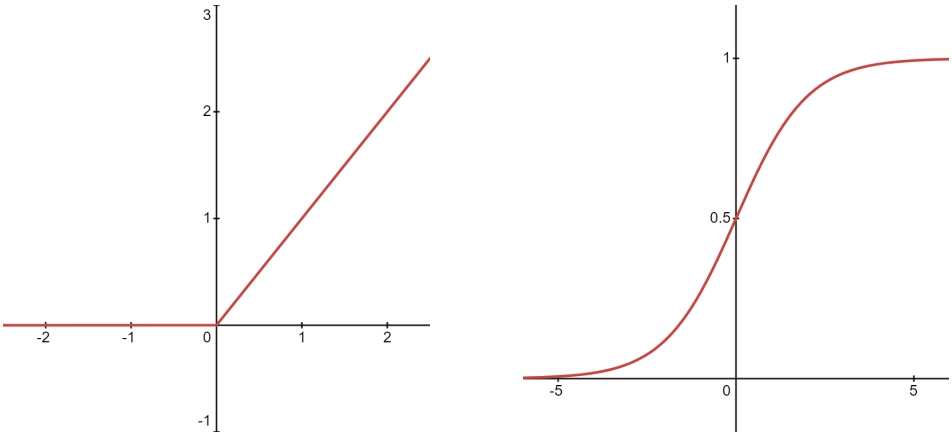
For an input a, common choices of activation functions include the *Rectified Linear Unit* (ReLU) [7], defined as

$$\text{ReLU}(a) = \max(0, a) \tag{2.7}$$

and the standard *logistic sigmoid* [5] (subsequently referred to as the logistic sigmoid), defined as

$$\sigma(a) = \frac{1}{1 + exp(-a)}. \tag{2.8}$$

Their graphical representations are provided in Figure 2.2a and 2.2b respectively.



(a) Rectified Linear Unit. **(b)** Logistic sigmoid.

Figure 2.2: Graphical representation of the Rectified Linear Unit and Logistic Sigmoid activation functions. The axes are scaled differently to better highlight the important features of each function.

2.3 Layers

2.3.1 Fully Connected Layers

The **Fully Connected Layer**, also referred to as the **Dense** layer, is one of the more commonly used layer types [18]. For N inputs, the jth output is calculated as

$$y_j = \sum_{i=1}^{N} x_i w_{ij} + b \tag{2.9}$$

where y_j is the j:th output, x_i the ith input, b a bias term independent of the input and w_{ij} the weight between x_i and y_j [10]. A fully connected layer is shown in Figure 2.3.

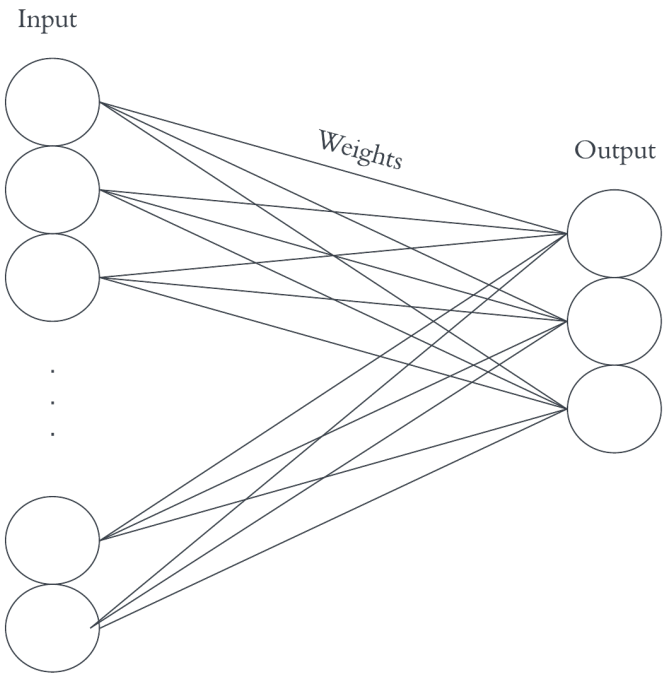


Figure 2.3: An example of a fully connected layer.

2.3.2 Convolutional Layers

Convolutional Layers specialize in capturing features for grid-like input data such as images or time series by employing the mathematical convolution operation [12]. While fully connected layers have weights between each input and output, convolutional layers leverage a multidimensional array, referred to as the *kernel* or *filter*, of weights which is repeatedly multiplied element-wise with the input and summed together. A visualization of this is provided in Figure 2.4.

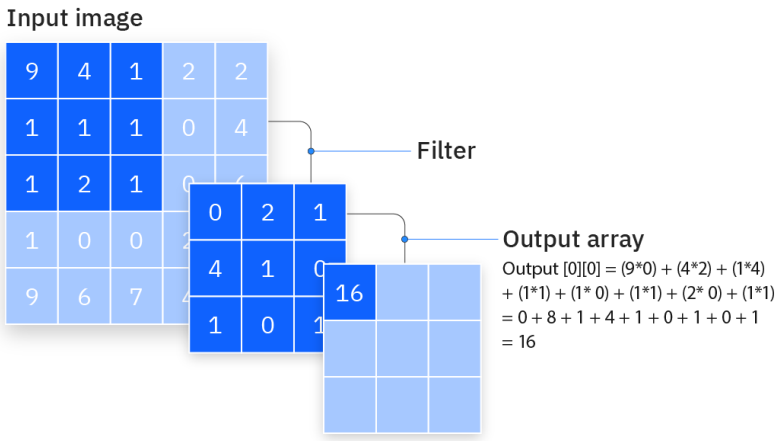


Figure 2.4: Example of a convolutional layer calculation [29].

The convolutional layer provides some useful features. The kernel size is typically small and every filter is reused for each position of the input [29]. Therefore parameter storage requirements are much lower than for fully connected layers and the reuse of each filter across all positions works as a form of regularization called *parameter sharing*. Convolutional layers possess another property referred to as *translation equivariance* which means that any shift in the input results in an equal shift in the output [16]. If followed by an activation function that is independent on the input position, this property can make the output invariant to translations. Translation invariance is convenient, especially for object classification [7], since the object will be classified the same regardless of its position in the input.

A neural network featuring mainly convolutional layers is often referred to as a *Convolutional Neural Network*, abbreviated as *CNN*.

2.3.3 Pooling Layers

Pooling layers capture a group of outputs and reduce them into single values using an aggregating function [8]. The layer can use any aggregating function but common choices are maximum and average functions [29]. Max pooling keeps only the highest value in its group as output while average pooling outputs a (possibly weighted) average of the group outputs. Visual examples of these are provided in Figure 2.5. Pooling layers are often used in conjunction with convolutional layers as a way of achieving translation invariance (see Section 2.3.2). It also reduces the spatial dimensions of the feature map that the convolutional layer provides [27], which helps reduce parameter requirements and increase computation speed [27].

Note that while average pooling is linear, max pooling is not. One might deem using a non-linear activation function unnecessary when using max pooling but, as highlighted by [20], performance is better when using a non-linear activation, even though max pooling is used. Hence, despite max pooling being non-linear, this is not the reason for its usage but rather a mere coincidence and it is not a

replacement for the non-linearity established by an activation function.

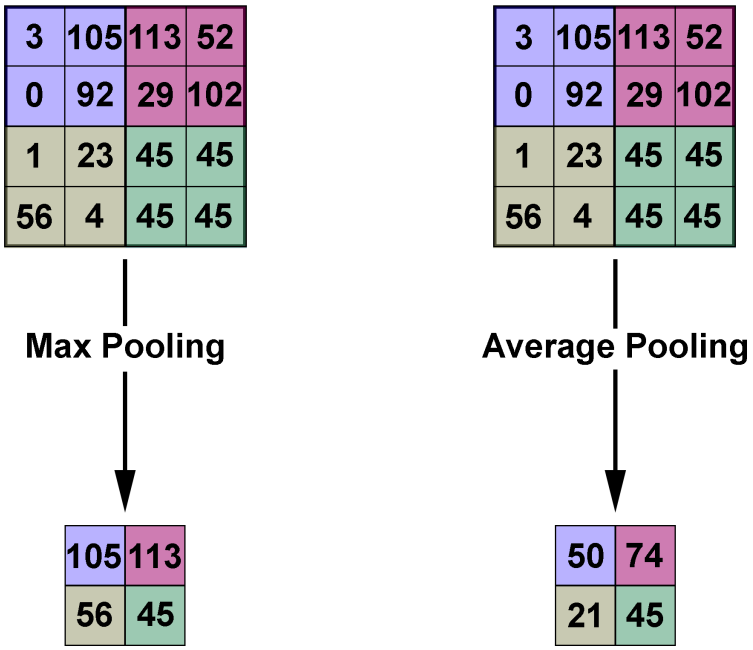


Figure 2.5: A visual representation of the non-linear max (left) and linear average (right) pooling operations.

2.3.4 Recurrent Layers

Recurrent layers consider past inputs when calculating current output, making them well suited for sequential input data [31]. The most general variant of a recurrent layer is the Fully Recurrent Layer [6]. In this definition, each node considers a hidden state, based on the calculations for the previous output, and the current input, producing a new hidden state and the current output. Mathematically, for a timestep t, these *update equations* are defined as

$$h_t = f(W_i x_t + W_h h_{t-1} + b_h) \tag{2.10}$$
$$y_t = f(W_o h_t + b_o). \tag{2.11}$$

In the equations above (2.10 and 2.11), h_t and h_{t-1} denote the current and previous hidden state, x_t the current input and y_t the corresponding output. W_i,

W_h, and W_o are weight matrices and b_h and b_o are bias terms. The subscripts i, h and o denote whether the weight or bias value is related to the input, hidden, or output part of the architecture respectively. f represents an activation function, shared by all nodes in the layer. This architecture is displayed as a block diagram in Figure 2.6.

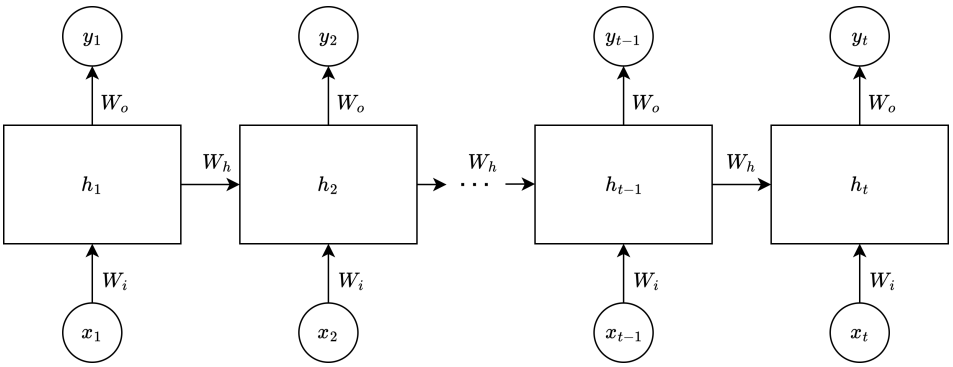


Figure 2.6: Block diagram of a traditional recurrent layer. Bias is omitted for improved readability. Inspired by Figure 1 in [31].

This thesis utilizes one specific recurrent layer, the Gated Recurrent Unit (GRU). Gated Recurrent Layers use control signals to weigh their components differently depending on the current state and input [12], which has proven to help derive long-term state dependencies better than traditional recurrent layers [31]. GRUs specifically, use two signals: the update signal u and the reset signal r. The update signal controls the trade-off between keeping the previous output and replacing it with the new output whereas the reset signal determines how the current output is used when calculating the next output.

The update equations for the GRU differ from the corresponding equations for the fully recurrent layer (see Equations 2.10 and 2.11). Two additional equations are required, one for each gating signal, and the current hidden state is dependent directly on the previous output y_{t-1} rather than the previous hidden state h_{t-1}. The resulting update process for the GRU is defined as follows

$$r_t = \sigma(W_{ir}x_t + W_{or}y_{t-1} + b_r) \tag{2.12}$$
$$u_t = \sigma(W_{iu}x_t + W_{ou}y_{t-1} + b_u) \tag{2.13}$$
$$h_t = f(W_{ih}x_t + W_{oh}r_t y_{t-1} + b_h) \tag{2.14}$$
$$y_t = u_t y_{t-1} + (1 - u_t)h_t. \tag{2.15}$$

Here W_{ir}, W_{iu} and W_{ih} are the weights of the connections between the input and their corresponding second subscript (reset, update, and hidden respectively) and W_{or}, W_{ou} and W_{oh} the weights for the corresponding output connections. Similarly; b_r, b_u, and b_h are the biases for the reset, update, and hidden values. σ is the *logistic sigmoid* activation function (see Equation 2.8) and f the output

activation function of the GRU. These calculations are illustrated in the block diagram in Figure 2.7. [31]

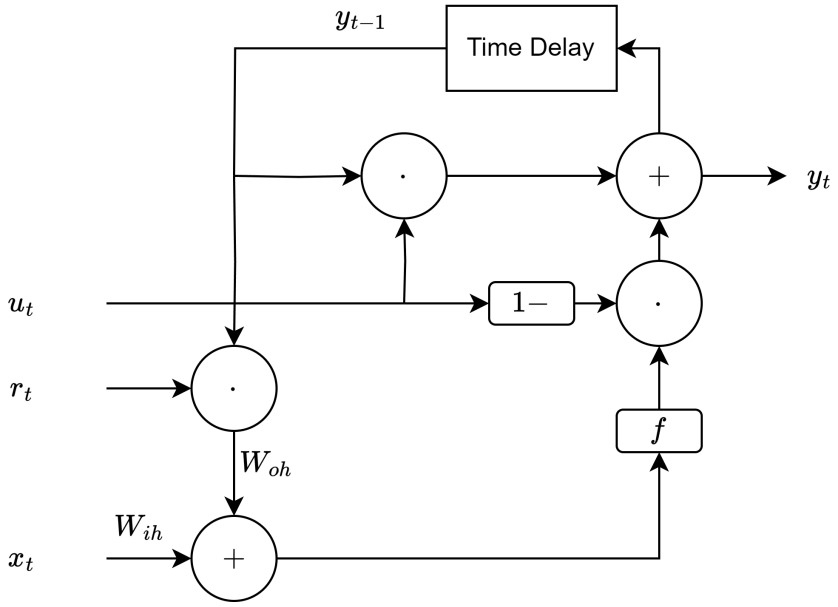


Figure 2.7: Block diagram of a GRU cell. Bias and update equations for gating signals omitted for brevity.

2.3.5 Batch Normalization

Batch normalization provides a novel way of breaking higher-order dependencies between layers that often occur in deeper networks. Layer weights are often dependent on values from other layers and, especially for deeper networks, these dependencies tend to be of a rather high order. Thus, when updating its weights, a layer assumes all other layers to remain unchanged. In practice, however, all layers are updated simultaneously. The inability to detect these higher-order dependencies introduces the risk of promoting poor-performing weight changes. Batch normalization counteracts this problem by normalizing small batches of output from the previous layer, leaving subsequent layers to operate on the normalized values instead. For a small batch of output vectors $\boldsymbol{H}$, the batch normalization output $\boldsymbol{H}'$ will be

$$\boldsymbol{H}' = \frac{\boldsymbol{H} - \boldsymbol{\mu}}{\boldsymbol{\sigma}}, \tag{2.16}$$

where $\boldsymbol{\mu}$ is the vector of mean values for each output vector and $\boldsymbol{\sigma}$ is the corresponding vector of standard deviations. The values of $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are included in the weight update process to discourage weight changes that increase the standard deviation or mean of the values in $\boldsymbol{H}$. To allow for testing individual samples, the batch values of $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are replaced with running averages. [12]

Chapter 3

Frequency Modulate Continuous Wave (FMCW) Radar

This thesis uses radar devices that employ a specific modern radar design technology, the Frequency Modulated Continuous Wave (FMCW) waveform. FMCW offers long-range detection and provides accurate range resolution with a low emitting power [9]. It does this by repeatedly emitting *chirps*. A chirp is a sinusoidal wave whose frequency increases linearly with time [25], as shown in the plots of Figure 3.1 and 3.2. The frequency-time plot (see Figure 3.2) is defined by its starting frequency f_0 (Hz), bandwidth B (Hz), and duration T_c (s). Its slope is denoted as S.

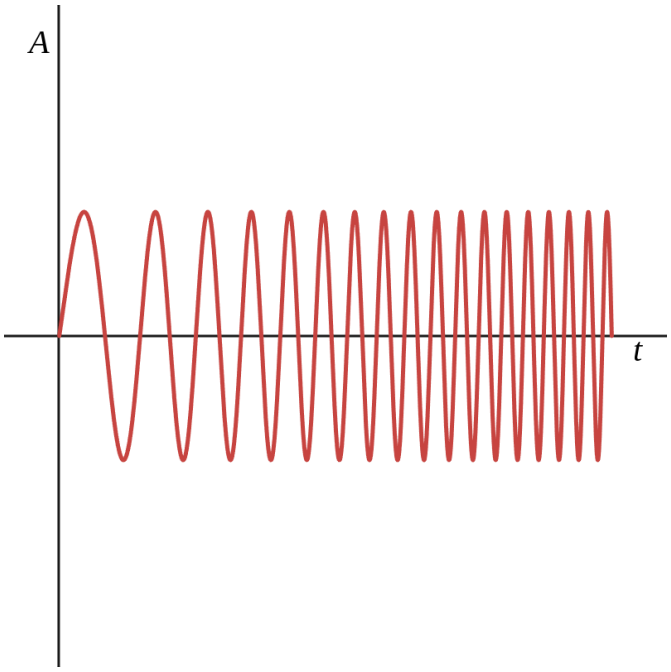


Figure 3.1: Amplitude-Time plot of a single linear chirp.

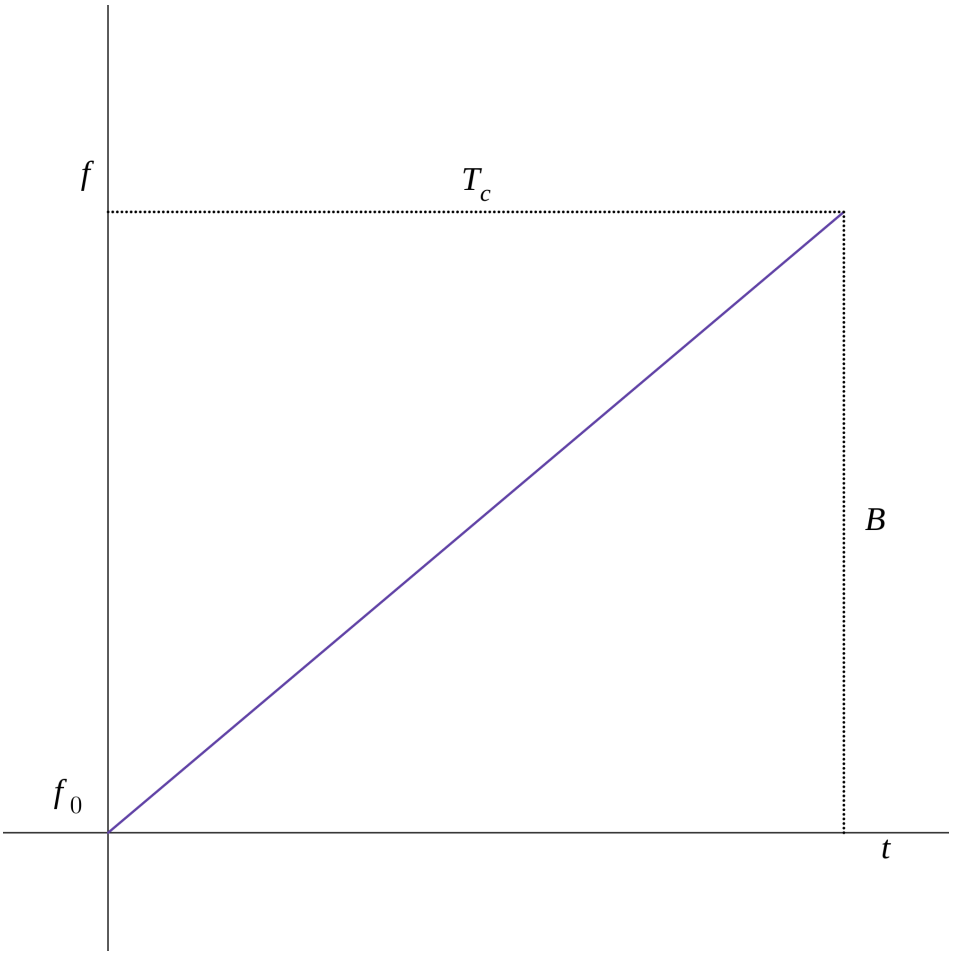


Figure 3.2: Frequency-Time plot of a single linear chirp with bandwidth B and duration T_c. Note that the y-axis starts at an initial frequency f_0.

3.1 Distance estimation

Many useful features can be extracted from the emitted chirps reflecting off an object, one of these being the distance to the object. As the synthesized analog chirp signal, transmitted by a transmitter (TX) antenna on the radar, is reflected off an object, the radar receiver (RX) antenna registers the reflection. The RX signal and TX signals are passed to a mixer device which outputs a sinusoid with a resulting frequency equal to the difference in frequency between its input signals [25]. Seeing as the RX signal is simply a delayed TX signal, the mixer output, referred to as the intermediate frequency (IF) signal, will have a constant frequency. This frequency is derived by measuring the round trip time τ (s) and multiplying it by the slope of the chirp, denoted S. The IF signal calculation is

shown in figure 3.3.

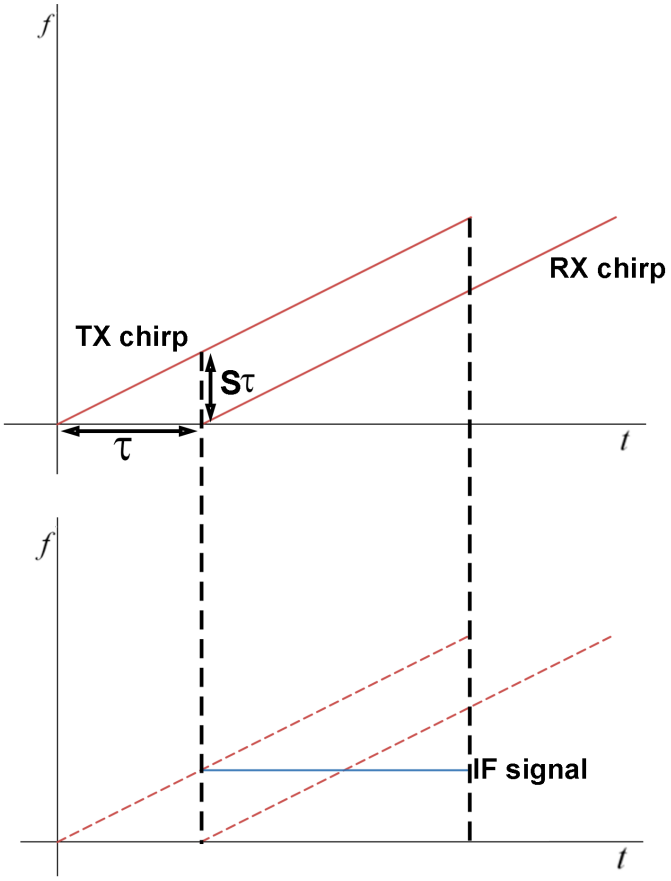


Figure 3.3: Deriving the Intermediate Frequency signal.

Since it takes the chirp a duration of τ to travel twice the distance between the radar and the object causing the reflection, the distance to the object d (m) can be deduced as

$$\tau c = 2d \implies d = \frac{\tau c}{2} \tag{3.1}$$

where c (m/s) is the speed of light.

3.2 Velocity estimation

By expanding on the principles of distance estimation, the velocity of the object reflecting the chirps can be obtained. At distance d from the radar unit, i.e. at

the point of reflection, the IF signal will be a sinusoid, which corresponds to

$$IF = A\sin\left(2\pi f t + \phi_0\right) \quad \text{where } f = \frac{S2d}{c} \tag{3.2}$$

for some phase ϕ_0 (rad/s). The phase ϕ_0 is very sensitive to small changes in round-trip time τ and two IF signals from the same object may exhibit large differences in phase due to these small shifts in round-trip time [25]. A round-trip time delay of $\Delta\tau$ results in a phase shift $\Delta\phi$

$$\Delta\phi = 2\pi\frac{\Delta\tau}{T} = 2\pi\frac{\Delta\tau}{\frac{1}{f}} = 2\pi f\Delta\tau \tag{3.3}$$

where T (s) is the period of the IF signal. Furthermore, note that a shift in distance d_0 to d_1, can be expressed as

$$\Delta\tau = \frac{2d_1}{c} - \frac{2d_0}{c} = \frac{2}{c}(d_1 - d_0) = \frac{2\Delta d}{c} \tag{3.4}$$

using Equation 3.1. Applying this to Equation 3.3 yields

$$2\pi f\Delta\tau = 2\pi\frac{c}{\lambda}\frac{2}{c}\Delta d = \frac{4\pi\Delta d}{\lambda}. \tag{3.5}$$

Hence, two chirps transmitted T_c seconds apart, reflected by the same object, will differ in distance according to

$$\Delta d = vT_c, \tag{3.6}$$

and using Equation 3.5, the velocity of this object can be inferred from the phase as

$$\Delta\phi = \frac{4\pi\Delta d}{\lambda} = \frac{4\pi vT_c}{\lambda} \implies v = \frac{\lambda\Delta\phi}{4\pi T_c}. \tag{3.7}$$

3.3 Angle estimation

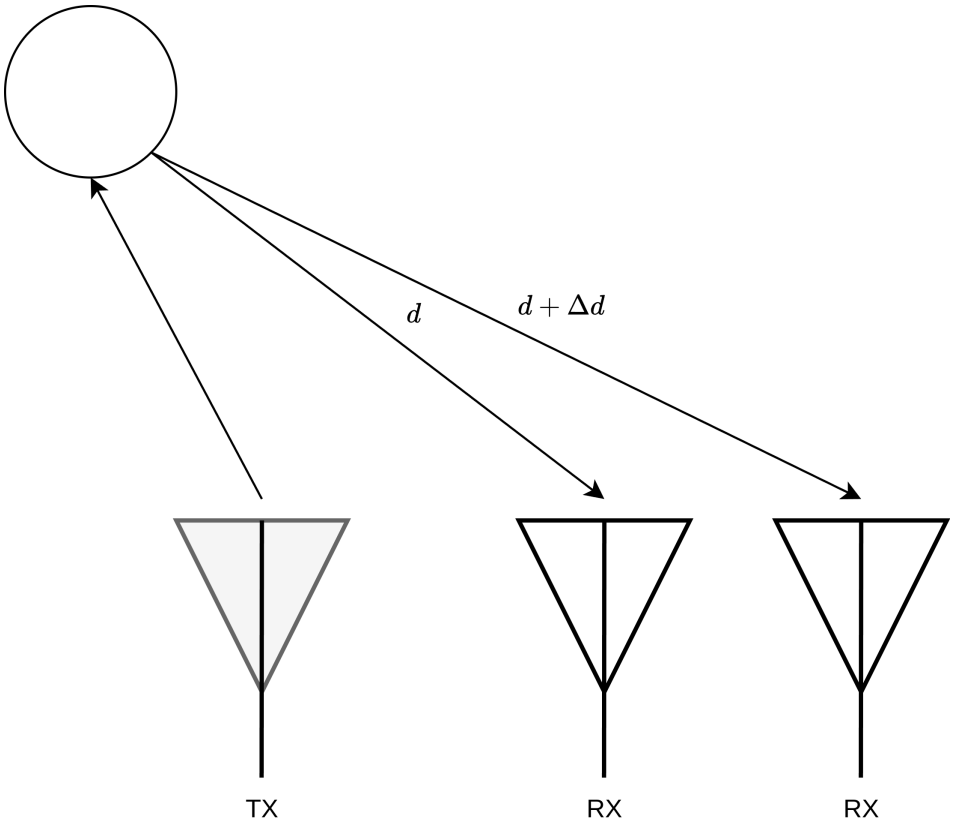


Figure 3.4: Two receiver (RX) antenna setup for angle estimation. Figure reconstructed from module 5 of [25].

Estimating the angle of arrival θ (rad) shares many similarities with the velocity approximation. However, angle estimation requires at least a two RX antenna setup according to Figure 3.4. Recall that the phase ϕ changes with the distance according to Equation 3.5. Similarly, two RX antennas spaced Δd meters apart produce a phase change

$$\Delta\phi = \frac{2\pi\Delta d_\theta}{\lambda} \quad \text{where } \Delta d_\theta = \Delta d \sin\theta. \tag{3.8}$$

Note the difference of a factor 2 compared to Equation 3.5 seeing as the original calculation refers to the round-trip distance whereas this Equation does not. The derivation of the relation $\Delta d_\theta = \Delta d \sin\theta$ is provided in Figure 3.5.

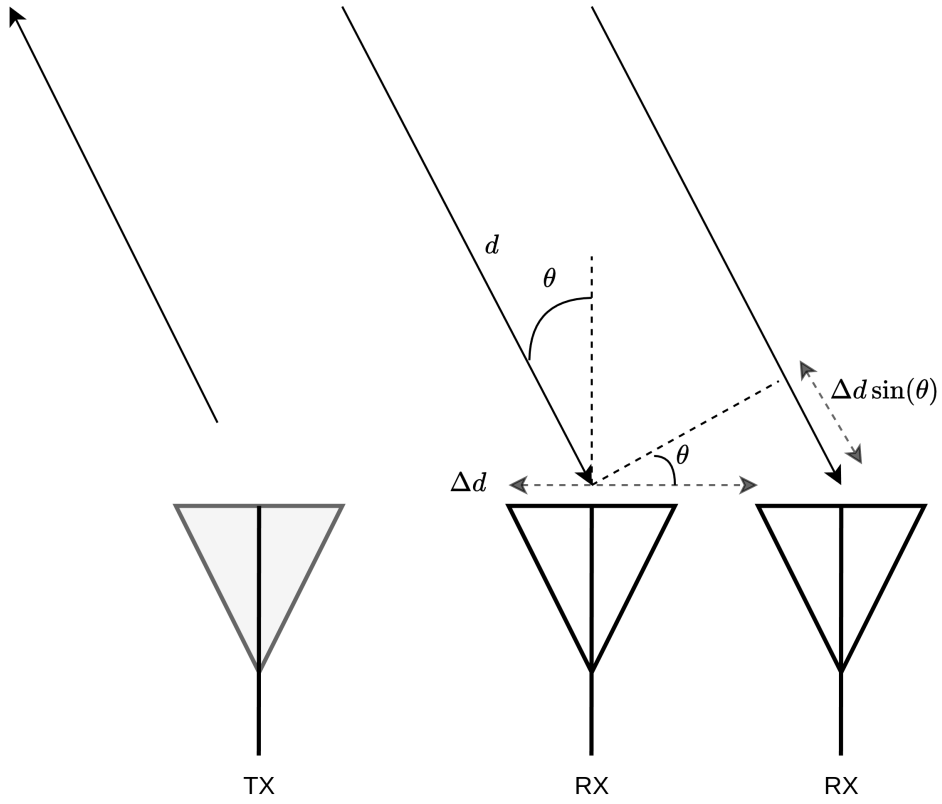


Figure 3.5: Derivation of *Angle of Arrival* (θ) using the two receiver (RX) antenna setup from Figure 3.4. Figure reconstructed from module 5 of [25].

Chapter 4
Data

4.1 Pre-processing

Before the radar data can be passed to the model for classification, it is converted into an appropriate representation through a series of pre-processing steps. This section aims to give a brief overview of these steps. A flowchart summarizing the entire pre-processing chain is provided in Figure 4.1.

The initial step includes all processes pertaining to signal processing. The raw radar data is first converted to a digital representation using an analog-to-digital converter (ADC). This digital data is passed through a Fast Fourier Transform (FFT), to extract the phase and amplitude of the input. Further, using the equations provided in Chapter 3, properties such as the radius, radial velocity, and angle are derived. Much of the data that belongs to the scenery rather than the target object can be removed by ignoring points with a velocity or signal strength below some predefined threshold.

What results from the signal processing step is a point cloud, assumed to only contain points related to the target object. These points are grouped by a clustering algorithm, which helps convert the data into a more structured representation of a single object, composed of points, rather than a set of individual point objects.

Lastly, this point cloud cluster is tracked over a series of radar inputs, sampled using a set interval. The tracking process yields what this thesis will consider a single unit of input data, a *radar track*.

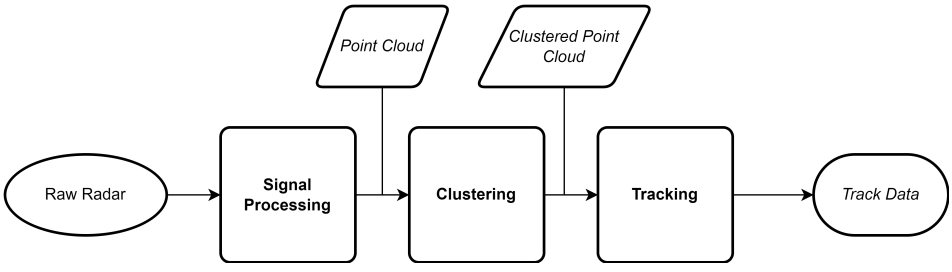


Figure 4.1: Summary of the data pre-processing pipeline.

4.2 Data Characteristics

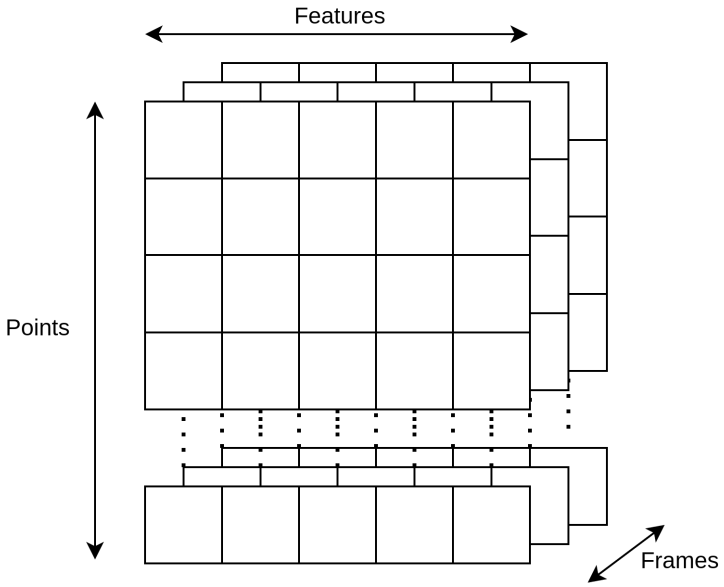


Figure 4.2: A visualization of the three-dimensional matrix describing a radar track.

Here, the important properties of the radar track are outlined to provide the reader with a greater insight into how the data is shaped and consequently justify the various design choices of the evaluated models.

The interval at which radar data is sampled during the tracking process (see Section 4.1) is determined by the radar design. Conventionally, this is expressed as a frame rate rather than a sampling interval, and consequently, a sample is referred to as a *frame*. Frame rates are measured in *frames per second* (FPS) rather than seconds. Hence, a radar with a sampling interval of for example 0.2s is subsequently referred to as having a frame rate of 5FPS.

Each frame in the radar track provides a clustered point cloud. A representation of the clustered point cloud is obtained by organizing the frame into a two-dimensional $N_p \times N_f$ matrix, for N_p points and N_f point features. Consequently, the full radar track can be shaped into a three-dimensional array by stacking several point cloud matrices. The result is visualized in Figure 4.2. A fixed number of frames (subsequently referred to as *window size*) are selected from each track to preserve a coherent input shape between tracks.

4.2.1 Points per frame

The pre-processing pipeline provides no guarantee about the number of points in a frame. On the contrary, it is common that the points per frame vary between

frames of a single radar track.

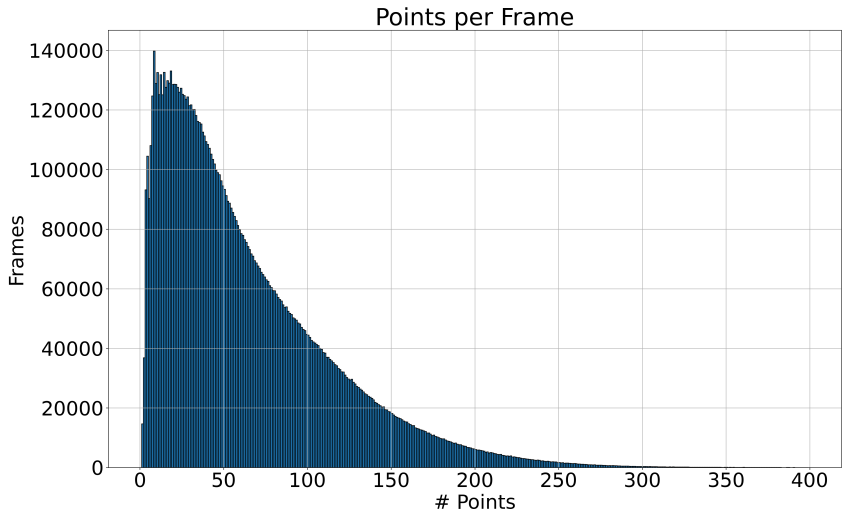


Figure 4.3: Distribution of points in a single frame over all frames in the dataset (Total of $11,163,134$ frames).

As per Figure 4.3, the distribution of points per frame is spread between 1 and 400, where the majority of all frames contain 100 points or less. For the model to be viable in practice, this number has to be fixed for all frames. Thus, for frames with fewer points there is a need to resample points until this number is reached whereas for frames with more than the required amount of points, only a subset of these are selected for the training process. Both these methods have drawbacks, however. Downsampling processes run the risk of removing points that might prove vital to the frame while upsampling might compromise the data by giving unnecessary importance to properties that do not contribute to the classification of the object in question.

4.2.2 Tracking

One of the more important features of the radar track is its tracking characteristic, which provides continuous measurements of the input rather than a single snapshot. This temporal aspect yields multiple views of a single target which should prove more robust to outliers than a single sample. It is also a higher-level feature than the likes of radial velocity and angle of arrival, which might benefit models with large capacity or depth.

4.2.3 Window size

The most obvious drawback with the temporal aspect of the data characteristics is the delay that comes with the implementation. As one of the dimensions of the

input data is a specified "window size", the tracking data of an object must be at least this many frames before the data can be used. For example, using a window size of 15 frames and a 2FPS radar would result in a delay of 7.5 seconds.

While a shorter window size is very much desired, so is the accuracy of the model. This problem of balancing is hard since a low accuracy with low latency is undesired, while a very high accuracy that requires that the subject is in the frame for a longer time might be unusable.

Chapter 5
Models

5.1 Baseline

The baseline model is described in the master's thesis [32] from 2021, by *N. Zandler Andersson*. It should be noted that this baseline was originally optimized for three classes.

5.2 Keras PointNet

The Keras PointNet model[1] is a version of the PointNet model, presented in [24]. Some adjustments to the pre-processing steps are made to accommodate the characteristics of the input data. Regarding network architecture, [24] proposes several fully connected layers in the initial components of the model, whereas the Keras version achieves the same functionality using convolutional layers with a kernel size of 1. This model has a very large capacity, approximately 10^6 parameters, depending on hyperparameter selection.

The Keras variant has three core components: a convolutional component, a fully connected component, and a custom component referred to as a "T-net". The T-net is itself a mini version of the PointNet model [24]. It is made up of a convolutional block, followed by a global max pooling function, and lastly, a fully connected block. Its output is calculated by computing the element-wise product of a transformation matrix, obtained by reshaping the fully connected output into an $N \times N$ matrix, and the original input. Two T-nets ($N = 3$ and $N = 32$ respectively) are interleaved with two blocks of CNNs. This is followed by a global max pooling function and a fully connected component. Figure 5.1 provides a visual representation of the architecture. The notion of CNN rather than a convolutional layer refers to Keras considering activation and normalization functions as layers as well. Each CNN is thus effectively a single, one-dimensional convolutional layer. All convolutional and fully connected layers use batch normalization and a ReLU activation function.

[1] Source code available at `https://github.com/keras-team/keras-io/blob/master/examples/vision/pointnet.py`

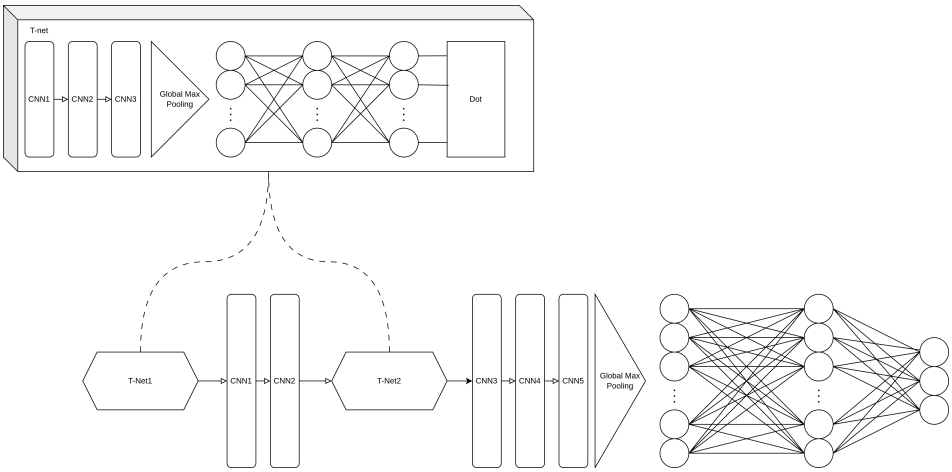


Figure 5.1: Overview of Keras PointNet architecture. Figure inspired by Figure 2 in [24].

5.3 Radar PointNet

The Radar PointNet model is, like Keras PointNet (see Section 5.2), a variant of the proposed architecture in [24], developed at Axis. While Keras PointNet is rather true to the actual PointNet, Radar PointNet is a much smaller network (approximately 10^3 parameters), with a few important structural changes. Both T-nets are removed and the convolutional component uses two rather than five convolutional layers. The first fully connected layer after the global max pooling function is replaced by a GRU layer to capture important properties of the temporal aspect present in the track. Convolutional and fully connected layers still utilize batch normalization and the ReLU activation function. The full network architecture is provided in Figure 5.2.

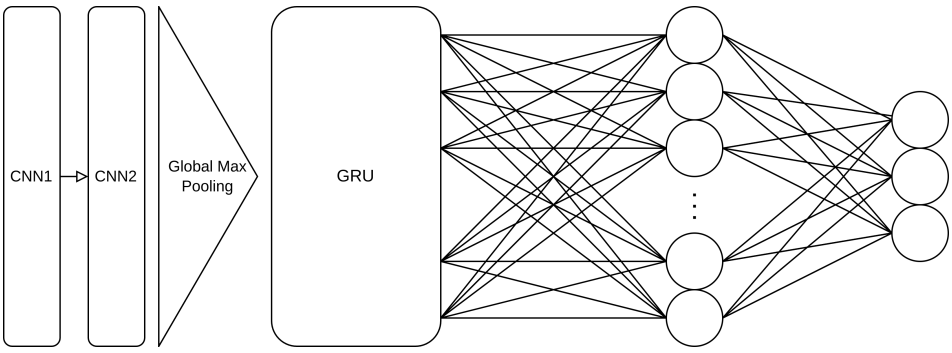


Figure 5.2: Overview of the Radar PointNet architecture.

Chapter 6
Method and Results

6.1 Model Variants

The six model variants which are evaluated against the baseline are summarized in Table 6.1. First, we use the plain Keras and Radar PointNet models, acting as baselines for their respective architectures. Since Keras PointNet and Radar PointNet share many similarities in architecture except for the T-net, we implement two variants of the Keras architecture with one or both T-nets removed, respectively. Similarly, we employ a Radar PointNet architecture where the GRU has been replaced with an additional CNN. Furthermore, as the Keras architecture has a much higher parameter count than the Radar architecture, a smaller Keras version is used to compare the architectures when the magnitude of parameters is similar.

Table 6.1: Descriptions of model variants evaluated.

Name	**Description**
RadarPointnet	Original Radar model (see Section 5.3)
RadarCNN	GRU replaced with CNN
KerasPointnet	Original Keras model (see Section 5.2)
Keras1Tnet	First T-Net removed
KerasNoTnet	Both T-Nets removed
KerasSmall	Number of layer weights downscaled by a factor 8

6.2 Finding Optimal Hyperparameters

To find appropriate values for the hyperparameters we employ the **Optuna** [1] framework for hyperparameter optimization. Optuna allows for using algorithms like TPE [4], which sample hyperparameters independently, and algorithms like CMA-ES [13] and GP-BO [26], that take relations between hyperparameters into account when conducting the search. Furthermore, it uses the *Asynchronous Successive Halving Algorithm* (ASHA) [17] to terminate instances that display poor intermediate performance early.

Frameworks like Optuna greatly reduces the amount of manual work required to find suitable parameters. However, it should be noted that Optuna still requires its users to define the search space for each parameter. Consequently, the optimization performance is limited by the choice of search space. This search space must in turn be limited for the whole process to remain computationally feasible. Only the best trial, based on validation performance, is used for model comparisons.

6.3 Training and Evaluation

Each model is trained on 80% of a training set of 88k radar tracks, reserving the last 20% for validation, repeated 5 times for different training and validation data splits. We run this process for 10 sets of hyperparameters, configured by Optuna, and select the hyperparameters with the greatest average performance on the corresponding 5 validation splits. Model performance is measured by applying the model, using the selected hyperparameters, on a test set of 2.4k samples.

6.4 Results

This section presents the relevant performance metrics for all models presented in Section 5.

To give an initial indication of how the models compare, we first provide some of the more general metrics, such as the overall accuracy. These measurements are provided in table 6.2. Seeing as some models are very similar in performance, the number of trainable parameters and floating point operations (FLOPs) are also displayed here, highlighting potential trade-offs between resource use and performance.

Table 6.2: Accuracy and resource requirements for all models. The best results are highlighted in bold.

Model	Accuracy (%)	Number of parameters	FLOPs
Baseline Model	87.14	2038k	7.6M
RadarPointnet	91.85	**3k**	**1.7M**
RadarCNN	**91.97**	46k	1.8M
KerasPointnet	86.16	751k	170M
Keras1Tnet	83.17	416k	101M
KerasNoTnet	90.46	209k	52M
KerasSmall	91.11	12k	3.1M

The precision and recall metrics for human and vehicle are provided in table 6.3. We also present two variants of the average recall and precision: the raw (macro) average and a weighted average, scaled by the distribution of classes in the test set. In Appendix A.2, we supply the confusion matrices from which the table values are derived.

Table 6.3: Precision and recall for Human (H) and Vehicle (V). The best results are presented in bold.

Method	Precision				Recall			
	H	V	Macro	Weighted	H	V	Macro	Weighted
Baseline Model	0.73	0.94	0.83	0.88	0.84	0.89	0.86	0.87
RadarPointnet	0.81	**0.97**	**0.89**	**0.92**	**0.91**	0.92	**0.92**	**0.92**
RadarCNN	0.82	0.96	**0.89**	**0.92**	0.89	0.93	0.90	**0.92**
KerasPointnet	0.69	0.94	0.82	0.88	0.86	0.86	0.86	0.86
Keras1Tnet	0.66	0.91	0.78	0.84	0.77	0.86	0.81	0.83
KerasNoTnet	**0.85**	0.92	**0.89**	0.90	0.78	**0.95**	0.86	0.90
KerasSmall	0.84	0.94	**0.89**	0.91	0.82	0.94	0.88	0.91

In addition to evaluating the number of correct outputs, we also consider how confident the model is in its predictions. A prediction is considered confident if its output probability is above some threshold, defined ahead of time. Table 6.4 presents the fraction of confident outputs each model provides, in the context of correct predictions and incorrect predictions.

Table 6.4: Fraction of confident predictions for correct (C) and incorrect (I) samples. The best results of each column are shown in bold.

Threshold / Method	0.7		0.8		0.9		0.95	
	C	I	C	I	C	I	C	I
Baseline Model	**0.97**	0.84	**0.95**	0.73	**0.91**	0.55	**0.84**	0.36
RadarPointnet	**0.97**	0.73	0.92	0.57	0.71	0.29	0.52	0.10
RadarCNN	0.95	0.68	0.83	0.46	0.60	0.17	0.55	0.13
KerasPointnet	0.92	**0.37**	0.86	**0.23**	0.69	**0.09**	0.54	**0.05**
Keras1Tnet	0.92	0.74	0.82	0.62	0.67	0.45	0.54	0.30
KerasNoTnet	0.96	0.70	0.91	0.59	0.82	0.37	0.71	0.24
KerasSmall	0.94	0.62	0.87	0.47	0.76	0.28	0.67	0.19

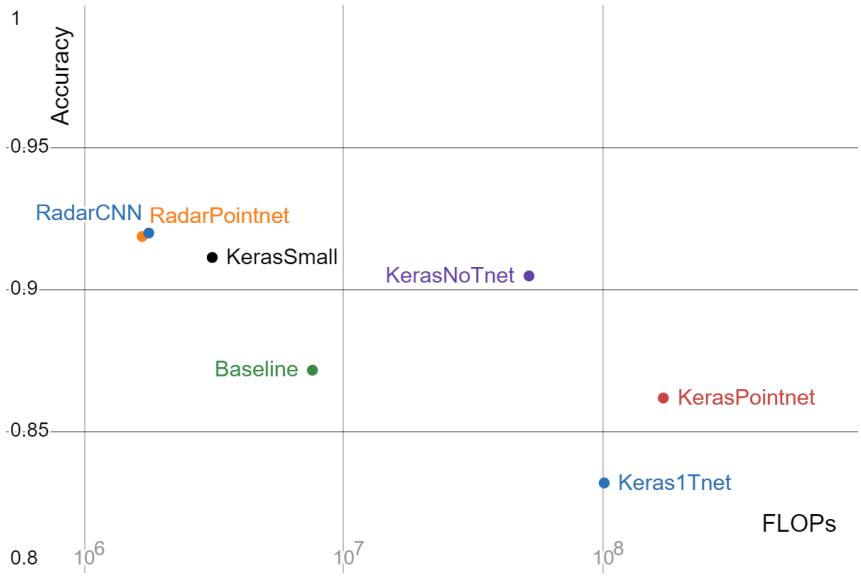


Figure 6.1: Accuracy and floating point operations of all models.

Figure 6.2 depicts how the accuracy of a model changes with different window sizes. It is, however, important to note that the hyperparameters used when generating Figure 6.2 were gathered from a hyperparameter optimization for a fixed window size.

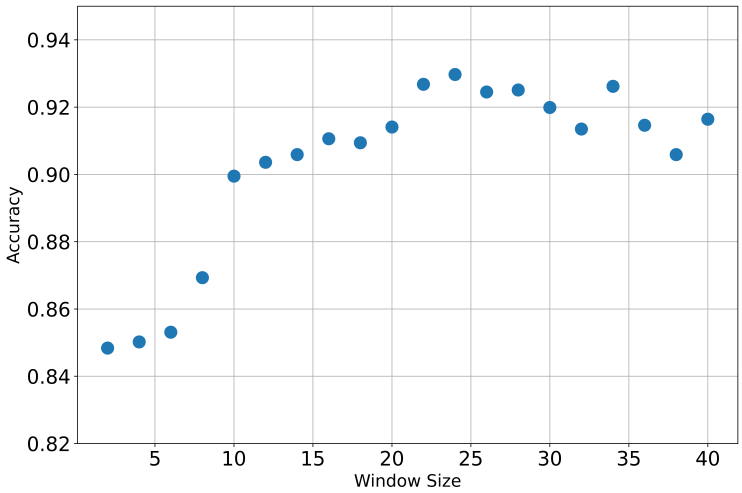


Figure 6.2: Accuracy of Radar PointNet model (see Section 5.3) with different window sizes.

Chapter 7
Discussion

7.1 Main Findings

Evaluating PointNet-based deep learning has provided an initial insight into its validity for radar track data applications. Both the Keras and Radar PointNet architectures show promising results, with three out of five contenders beating out the baseline model in overall accuracy of the test set (see Table 6.2). We have found that varying the Keras architecture impacts performance greatly and identified several configurations where performance is worse than the baseline. While some of our results suggest that the T-net (see Section 5.2) module in the Keras model decreases performance, the high accuracy of KerasSmall indicates that the root cause of the poor performance might not be related to the T-net configuration, but rather the overhead of parameters added by including it in the network. We have concluded that the RadarPointnet model exhibits the best performance, improving upon the overall accuracy of the baseline model by 4.22 percentage points. It also accomplishes this using far fewer parameters than the other models. While KerasSmall provides a similar magnitude of parameters, RadarPointnet outperforms KerasSmall by 0.74 percentage points of overall accuracy.

From Table 6.3 we have noted that, except for KerasNoTnet, all models exhibit much greater difference in precision between human and vehicle than the corresponding values for recall. This indicates that the models tend to misclassify vehicle samples as human. Seeing as vehicles generally provide a greater area that can reflect chirps, we believe that a poorly sampled vehicle could exhibit features closely resembling that of the average human sample.

Furthermore, we observed a negative correlation between the performance of our models and trainable parameters and FLOPS (see Table 6.2 and Figure 6.1) which merits further speculation. The top 3 models with regard to overall accuracy achieve this performance using far fewer parameters and FLOPS than their competitors. This might be related to the sparsity of the radar data.

A low parameter count is a prerequisite for deploying a model on a device, seeing as resource availability can be rather limited. This advocates for RadarPointnet, RadarCNN, or KerasSmall being possible candidates for replacing the baseline considering on-device classification, seeing as they outperform the baseline model in both resource requirement and accuracy.

7.2 Confidence and Thresholding

In Table 6.4, we introduce the confidence of predictions from all models, at different thresholds. All models retain higher confidence in their correct predictions than their incorrect ones but diverge differently when the threshold increases. We found that the baseline model retains high trust in all of its predictions, including the incorrect ones. While confidence in incorrect predictions is undesirable, the impact of this metric decreases as model accuracy is improved, seeing as the total number of incorrect predictions is lower. We noted that the Keras and Radar models reach promising confidence at the threshold of 0.8, decreasing their confidence of incorrect prediction substantially whilst preserving confidence in the range of $[0.82, 0.91]$ for correct predictions. For higher thresholds, we concluded that the steep decrease in correct prediction confidence outweighs the performance gain in weakening the incorrect prediction confidence further.

In a real-world scenario, the confidence metric helps give additional feedback about the reliability of a model. The area of surveillance generally demands a higher guarantee of robustness in its resources than other fields, seeing as malfunctions can be detrimental to the user [21]. We have found that using confidence in conjunction with accuracy provides more information about the reliability of the classifier than using any of these metrics individually.

7.3 Future Work

This work shows multiple potential deep learning classifiers that prove proficient in the field of radar-based point cloud classification and while we believe these insights to be notable additions to the field, there are many characteristics of these classifiers left to explore. One such characteristic is the window size. As briefly discussed in Section 4.2, we use a fixed window size for all models. Varying the window size and investigating its impact on the confidence and accuracy metrics is of great interest for determining whether a model is suitable for running on-device, seeing as the window size directly influences the delay of the classification output. Presented in Figure 6.2, there seems to be a correlation, where the accuracy increases by some percentage points for every additional frame at first, but this increase stagnates after some 10 to 20 frames. This does suggest that a good window size is somewhere in this range of frames and should be an optimal trade-off between delay, in terms of how many frames the model requires, and accuracy. However, a more thorough hyperparameter optimization for all window sizes could prove a different range of window sizes to be optimal. Another possible approach could be defining a method for processing partial inputs, e.g. by padding tracks shorter than the window size. While the delay can be reduced to a great extent using this technique, the inherent delay from pre-processing the raw radar data prevents the delay from being eliminated. Furthermore, padding would require reevaluating the structure of the model to account for this new aspect.

While we note the importance of considering not only accuracy but also confidence, determining the relation of these two metrics is outside the scope of this thesis. Appendix A.1 displays the fraction of confident correct and confident incorrect predictions left as the threshold increases.

Furthermore, the hyperparameter optimization performed in this study is not very extensive. Given the inherently large search space of hyperparameters and the scope of our thesis, we have intentionally limited the amount of work on this optimization. While substantial enough to indicate how models compare and perform, a real-world application of these techniques would most likely warrant further optimization of the hyperparameters.

References

[1] T. Akiba et al. “Optuna: A Next-generation Hyperparameter Optimization Framework”. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’19. Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 2623–2631. ISBN: 9781450362016. DOI: 10.1145/3292500.3330701. URL: https://doi.org/10.1145/3292500.3330701.

[2] J. Bai et al. “Traffic participants classification based on 3D radio detection and ranging point clouds”. en. In: *IET Radar, Sonar & Navigation* 16.2 (Feb. 2022), pp. 278–290. ISSN: 1751-8784, 1751-8792. DOI: 10.1049/rsn2.12182. URL: https://onlinelibrary.wiley.com/doi/10.1049/rsn2.12182 (visited on 06/04/2024).

[3] G. Baudic et al. “14 - Using emulation to validate applications on opportunistic networks”. In: *Advances in Delay-Tolerant Networks (DTNs) (Second Edition)*. Ed. by J. P. C. Rodrigues. Second Edition. Woodhead Publishing Series in Electronic and Optical Materials. Woodhead Publishing, 2021, pp. 273–280. ISBN: 978-0-08-102793-6. DOI: https://doi.org/10.1016/B978-0-08-102793-6.00014-X. URL: https://www.sciencedirect.com/science/article/pii/B978008102793600014X.

[4] J. Bergstra et al. “Algorithms for Hyper-Parameter Optimization”. In: *Advances in Neural Information Processing Systems*. Ed. by J. Shawe-Taylor et al. Vol. 24. Curran Associates, Inc., 2011. URL: https://proceedings.neurips.cc/paper_files/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf.

[5] C. M. Bishop. *Pattern recognition and machine learning*. Information science and statistics. New York: Springer, 2006. ISBN: 9780387310732.

[6] J. A. Bullinaria. “Recurrent neural networks”. In: *Neural Computation: Lecture* 12.1 (2013).

[7] *Deep Learning in a Nutshell: Core Concepts*. en-US. Nov. 2015. URL: https://developer.nvidia.com/blog/deep-learning-nutshell-core-concepts/ (visited on 04/05/2024).

[8] M. Deprez and E. C. Robinson. "Chapter 11 - Convolutional neural networks". In: *Machine Learning for Biomedical Applications*. Ed. by M. Deprez and E. C. Robinson. Academic Press, 2024, pp. 233–270. ISBN: 978-0-12-822904-0. DOI: https://doi.org/10.1016/B978-0-12-822904-0.00016-9. URL: https://www.sciencedirect.com/science/article/pii/B9780128229040000169.

[9] *Design of new Frequency Modulated Continuous Wave (FMCW) target tracking radar with digital beamforming tracking – A scientific report on high fidelity radar electronic warfare modelling and simulation and hardware in-the-loop*. Dec. 2019. URL: https://pubs.drdc-rddc.gc.ca/BASIS/pcandid/www/engpub/DDW?W%3DSYSNUM=811367&r=0 (visited on 05/09/2024).

[10] A. D. Dongare, R. R. Kharde, A. D. Kachare, et al. "Introduction to artificial neural network". In: *International Journal of Engineering and Innovative Technology (IJEIT)* 2.1 (2012), pp. 189–194.

[11] Z. Feng et al. "Point Cloud Segmentation with a High-Resolution Automotive Radar". In: *AmE 2019 - Automotive meets Electronics; 10th GMM-Symposium*. 2019, pp. 1–5.

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

[13] N. Hansen and A. Ostermeier. "Completely Derandomized Self-Adaptation in Evolution Strategies". In: *Evolutionary Computation* 9.2 (2001), pp. 159–195. DOI: 10.1162/106365601750190398.

[14] *IBM Developer*. URL: %5Curl%7Bhttps://developer.ibm.com/learningpaths/get-started-with-deep-learning/an-introduction-to-deep-learning/%7D (visited on 04/15/2024).

[15] C. Janiesch, P. Zschech, and K. Heinrich. "Machine learning and deep learning". en. In: *Electronic Markets* 31.3 (Sept. 2021), pp. 685–695. ISSN: 1422-8890. DOI: 10.1007/s12525-021-00475-2. URL: https://doi.org/10.1007/s12525-021-00475-2 (visited on 04/05/2024).

[16] O. S. Kayhan and J. C. Gemert. "On Translation Invariance in CNNs: Convolutional Layers Can Exploit Absolute Spatial Location". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.

[17] L. Li et al. “A System for Massively Parallel Hyperparameter Tuning”. In: *Proceedings of Machine Learning and Systems*. Ed. by I. Dhillon, D. Papailiopoulos, and V. Sze. Vol. 2. 2020, pp. 230–246. URL: https://proceedings.mlsys.org/paper_files/paper/2020/file/a06f20b349c6cf09a6b171c71b88bbfc-Paper.pdf.

[18] *Linear/Fully-Connected Layers User's Guide*. en. URL: https://docs.nvidia.com/deeplearning/performance/dl-performance-fully-connected/index.html (visited on 04/09/2024).

[19] A. Luque et al. “The impact of class imbalance in classification performance metrics based on the binary confusion matrix”. In: *Pattern Recognition* 91 (2019), pp. 216–231. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2019.02.023. URL: https://www.sciencedirect.com/science/article/pii/S0031320319300950.

[20] D. Mishkin, N. Sergievskiy, and J. Matas. “Systematic evaluation of convolution neural network advances on the imagenet”. In: *Computer vision and image understanding* 161 (2017), pp. 11–19.

[21] CNN Newsource. *Texas high school goes into lockdown due to AI security system's false alarm*. en. Oct. 2023. URL: https://news4sanantonio.com/newsletter-daily/texas-high-school-goes-into-lockdown-due-to-ai-security-systems-false-alarm-students-friends-family-scared-security-system-campus (visited on 05/29/2024).

[22] J. Niemeyer, F. Rottensteiner, and U. Soergel. “Conditional Random Fields for LiDAR Point Cloud Classification in Complex Urban Areas”. In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* I-3 (2012), pp. 263–268. DOI: 10.5194/isprsannals-I-3-263-2012. URL: https://isprs-annals.copernicus.org/articles/I-3/263/2012/.

[23] P. Probst, A. Boulesteix, and B. Bischl. “Tunability: Importance of Hyperparameters of Machine Learning Algorithms”. In: *Journal of Machine Learning Research* 20.53 (2019), pp. 1–32. URL: http://jmlr.org/papers/v20/18-444.html.

[24] C. R. Qi et al. “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017.

[25] S. Rao. *Introduction to mmwave Sensing: FMCW Radars - Module 1: Range Estimation*. URL: https://www.ti.com/content/dam/videos/external-vide

os/en-us/2/3816841626001/5415203482001.mp4/subassets/mmwaveSensing-FMCW-offlineviewing_0.pdf (visited on 05/09/2024).

[26] B. Shahriari et al. "Taking the Human Out of the Loop: A Review of Bayesian Optimization". In: *Proceedings of the IEEE* 104.1 (2016), pp. 148–175. DOI: 10.1109/JPROC.2015.2494218.

[27] P. Singh, P. Raj, and V. P. Namboodiri. "EDS pooling layer". In: *Image and Vision Computing* 98 (2020), p. 103923. ISSN: 0262-8856. DOI: https://doi.org/10.1016/j.imavis.2020.103923. URL: https://www.sciencedirect.com/science/article/pii/S026288562030055X.

[28] O. Vinyals, S. Bengio, and M. Kudlur. "Order matters: Sequence to sequence for sets". In: *arXiv preprint arXiv:1511.06391* (2015).

[29] *What are Convolutional Neural Networks? | IBM*. en-us. URL: https://www.ibm.com/topics/convolutional-neural-networks (visited on 04/05/2024).

[30] N. Yastikli and Z. Cetin. "Classification of LiDAR Data with Point Based Classification Methods". In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLI-B3 (2016), pp. 441–445. DOI: 10.5194/isprs-archives-XLI-B3-441-2016. URL: https://isprs-archives.copernicus.org/articles/XLI-B3/441/2016/.

[31] F. Ye and J. Yang. "A Deep Neural Network Model for Speaker Identification". en. In: *Applied Sciences* 11.8 (Apr. 2021), p. 3603. ISSN: 2076-3417. DOI: 10.3390/app11083603. URL: https://www.mdpi.com/2076-3417/11/8/3603 (visited on 04/10/2024).

[32] N. Zandler Andersson. *Moving Target Classification with Radar Point-Clouds and Supervised Contrastive Learning*. 2021.

[33] J. Zhang, X. Lin, and X. Ning. "SVM-Based Classification of Segmented Airborne LiDAR Point Clouds in Urban Areas". In: *Remote Sensing* 5.8 (2013), pp. 3749–3775. ISSN: 2072-4292. DOI: 10.3390/rs5083749. URL: https://www.mdpi.com/2072-4292/5/8/3749.

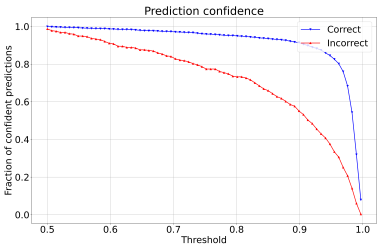
[34] Z. Zhao et al. "Point Cloud Features-Based Kernel SVM for Human-Vehicle Classification in Millimeter Wave Radar". In: *IEEE Access* 8 (2020), pp. 26012–26021. DOI: 10.1109/ACCESS.2020.2970533.

Appendix A

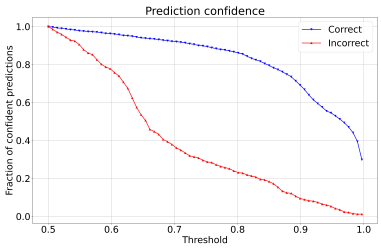
Appendix

A.1 Confidence Plots

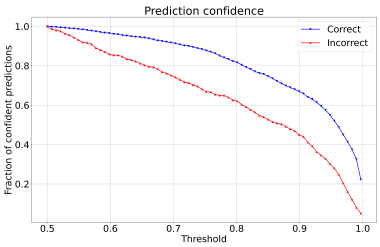
Here, we provide visualizations for the confidence of each model's prediction. Confidence values are sampled evenly in the threshold range of 0.5 to 1.0. The confidence values for correct predictions and incorrect predictions are processed separately.



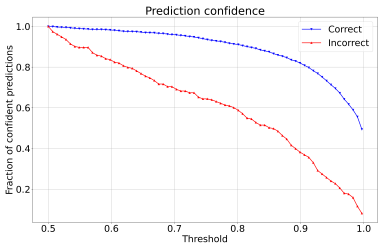
(a) Baseline.



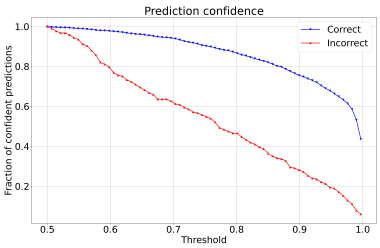
(b) KerasPointnet.



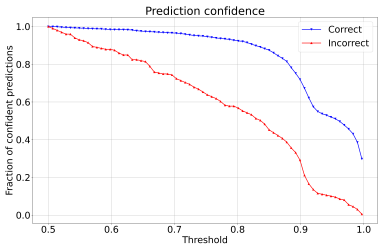
(c) Keras1Tnet.



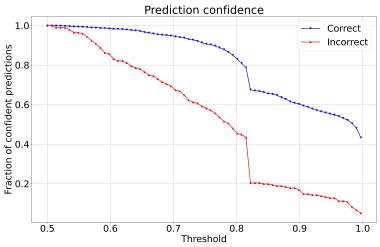
(d) KerasNoTnet.



(e) KerasSmall.



(f) RadarPointnet.



(g) RadarCNN.

Figure A.1: Fraction of confident predictions left as threshold increases.

A.2 Confusion Matrices

Figure A.2: Confusion Matrices.