CHALMERS
UNIVERSITY OF TECHNOLOGY

On Data-Driven Filter Development in Airborne Radar Systems

Investigating Feasibility and Verification of Machine Learning in Radar Signal Processing

Master's thesis in Systems, Control and Mechatronics

Emil Raudberget
Sanna Sandberg

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020

On Data-Driven Filter Development in Airborne Radar Systems

Investigating Feasibility and Verification of Machine Learning in Radar Signal Processing

EMIL RAUDBERGET
SANNA SANDBERG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Signal Processing and Biomedical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2020

On Data-Driven Filter Development in Airborne Radar Systems
Investigating Feasibility and Verification of Machine Learning in Radar Signal Processing
EMIL RAUDBERGET
SANNA SANDBERG

Supervisor: Miriam Fransson, Saab AB
Examiner: Tomas McKelvey, Chalmers University of Technology

Master's Thesis 2020
Department of Electrical Engineering
Division of Signal Processing and Biomedical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Data-driven fighter jets

Typeset in LaTeX
Gothenburg, Sweden 2020

Data-driven Filter Development in Airborne Radar Systems
Investigating Feasibility and Verification of Machine Learning in Radar Signal Processing
Emil Raudberget
Sanna Sandberg
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Machine learning has gained popularity in many fields due to its success in e.g. image analysis. Radar data can be represented as images and thus machine learning was expected to be useful in radar signal processing. Recent studies have shown promising results from machine learning in the field of radar applications, yet there are few machine learning implementations in real radar systems. This thesis aimed to study the feasibility of machine learning in airborne radar signal processing, by creating a simple radar target classification function, aimed at learning micro doppler signatures. A custom cross validation method was proposed that enabled detecting dataset bias, which was used together with a method from Explainable AI to discover that the network had learned nonuseful features.

To use machine learning in radar signal processing thus require a more sophisticated method for aggregating the dataset. This led to the conclusion that the greatest challenges in implementing a deep learning function in airborne radar systems are to collect an unbiased dataset and to verify that the network has learned useful features.

Keywords: airborne radar, machine learning, verification, signal processing, deep learning, neural networks, cross-validation, Explainable AI, data-driven, pruning.

Acknowledgements

The authors would like to show gratitude towards Miriam Fransson at Saab AB for supervising this work and providing great support and guidance. The authors would like to thank Tomas McKelvey, the examiner of this work, for his valuable advice and comments. Finally a lot of gratitude is directed towards Richard Nilsson and the people at Saab AB for their support and the opportunity to perfom this work at their facilities.

Emil Raudberget & Sanna Sandberg, Gothenburg, May 2020

Contents

List of Figures

List of Tables

List of Abbreviations

ANN	Artificial neural network
ATC	Air traffic control
CFAR	Constant false alarm rate
CV	Cross-validation
CPI	Coherent pulse interval
CNN	Convolutional neural network
GE	Generalization error
HRRP	High resolution range profile
ICAO	International Civil Aircraft Organization
LSTM	Long short-term memory
MCV	Metadata cross-validation
PDR	Pulse doppler radar
PRF	Pulse repetition frequency
PRI	Pulse repetition interval
RD-map	Range-doppler map
ReLU	Rectified linear unit
RNN	Recurrent neural network
STT	Single target track
TWS	Track while scan

1
Introduction

Filters in radar systems are designed to look for features in the data that the sensor captures. Building filters for an airborne radar system is expensive and the physical restrictions from a flying system further complicate the process. Since the Second World War airborne radar systems have been used to detect aerial targets [1]. However, the flying vessels of tomorrow are not known to anyone and as the speed of technological innovation increases so must development of airborne radar filters to remain relevant. As digitalization has increased the flow of data in society, data-driven development such as Artificial Intelligence (AI) is becoming a key driver for economic growth according to [2]. The expectation is thus that AI could contribute to producing a competitive airborne radar system on the modern market.

Data-driven development is producing promising results on radar data [3, 4, 5, 6, 7, 8], and advancements in modern computers enable implementing AI under the constrains of a flying system, proving its potential. A more rapid design process is also expected from using data-driven development in the radar system, which could facilitate the process of adding new functionality. However, industrialization of AI cannot occur without a secure process of verifying its performance. In additon, implementing a new technology will always involve obstacles, hence a couple of steps remains before it is possible to benefit from using AI in airborne radar. One of many challenge with data driven development does not only lie in collecting a great amount of data, but how to aggregate the information in a meaningful way. Earlier research has focused on synthetic data or scenarios not connected to the environment of an airborne radar system.

Therefore it is interesting to investigate the feasibility of machine learning in airborne radar systems by performing a small-scale test using real-world data, and to find an approach to verify its functionality.

1.1 Deep learning

Artificial intelligence is the practice of making technology mimic human behavior and intelligence. Machine learning is a field within AI and a technique to develop an algorithm without defining a set of complex rules, often implemented as learning a model from a dataset. Modern machine learning approaches often use *deep learning* as a method to learn this model from the dataset. Deep learning fits a function to patterns in the data using an *artifical neural network* (ANN), and is often considered

state-of-the-art technology for image analysis tasks [9].

During supervised learning the model fitting process is based upon example input-output pairs. In the case of image classification, the training dataset would consist of images with a corresponding label. The performance of the machine learning algorithm is thus a direct result of how well the training dataset represents the target environment and if there are useful patterns in the data to learn.

1.2 Radar

By transmitting and listening to radio waves the radar system can determine speed, direction and distance to a target, even in darkness and bad weather. Detecting and classifying aerial targets is a valuable capability and during air combat it is especially important to distinguish between targets. The radar echo can be represented as range-doppler maps (RD-maps), which are image like matrices showing the strength of the radar signal response with respect to range and range-rate. Different targets have different frequency profiles dependent on how the radio waves are reflected off the targets. Therefore targets can appear different in the RD-maps and are in some cases even distingusihable by the human eye. Range-doppler maps are an important analysis tool for a radar system, and enable the use of image analysis tools for radar data.

1.3 Problem description

As the name suggests, data-driven development is no better than the data itself. Verifying the performance of a data-driven algorithm requires knowledge of the dataset. This knowledge must provide insight into what meaningful data is, in other words, if there exist patterns in the data and if the patterns will teach the algorithm anything desirable. With that kind of knowledge the performance of the machine learning algorithm could be estimated for data not available during training and evaluation, or how to collect new data. Since machine learning does not require any set of rules for training it is not obvious what aspects in the dataset are important or what could impair the function. The ANN is a ”black-box” and it might find patterns in the data not comprehensible to humans, thus complicating the process of understanding what is behind its performance. Allowing it to learn from experience without providing any prior knowledge is a strength, however in order to trust the algorithm or to create a good dataset it must be possible to unravel what meaningful features are and understand what the deep learning algorithm considers useful.

The problem of understanding a data-driven algorithm and creating trust can be investigated in multiple ways. For example, by opening up the black box and understanding its behaviour, or by verifying that the dataset is close to the target environment, or performing tests with the fully trained algorithm to investigate its behaviour in different environments. The focus appears to be on the dataset and

not the network architecture.

Using real-world data positions the dataset close to the target environment, yet it is not certain to contain any meaningful patterns for a neural network to learn. How can the data collection method, including labelling, be designed such that an ANN learns useful features?

Since machine learning has been succesful on image analysis and radar data can be represented as images, target classification is a suitable function to develop within the timeline of this work in order to try to answer some of these questions. It is however unknown how feasible the use of RD-maps with real-world data is and what problems need to be solved before it is a feasible method. Investigating the potential of machine learning based only on classification performance could overlook other benefits in the design process, or obscure obstacles that could impair the system. For data driven development to take place in a complex radar system it needs to improve some aspect of the system, whether it be cost, computational complexity, development speed or performance. Hence focusing on the data and not the network architecture could provide knowledge which is useful for developing other functions with machine learning in the airbone radar system.

One big advantage of using machine learning is its capacity of "learning without rules", but then it might be percieved as contradictory to look into the black box of the ANN or trying to find these "5rules". In order to investigate the process of creating radar filters using machine learning, without bringing to much prior knowledge into the process, an interative trial-and-error approach could accumulate information of the task at hand without interfering with the principles of machine learning.

1.4 Purpose

This thesis aims to determine challenges that arise when developing a machine learning algorithm in airborne radar signal processing, and whether they can be overcome. If the challenges are possible to overcome, machine learning could be considered feasible in current airborne radar system.

1.5 Scope and limitations

To discover the challenges of implementing machine learning algorithms, a prototype function was developed using supervised deep learning. A small-scale implementation of the full development process was evaluated. This thesis focuses on the signal processing part of data driven development. There was no actual implementation of the function in a real system. The intended functionality was to distinguish target classes in the radar signal using RD-maps.

The main focus of this thesis lie in the data aspect and the necessary work before using deep learning in the system. Thus achieving the best classification accuracies was not prioritised. The neural network architecture remained the same throughout the work and instead the dataset was altered to obtain the results. The network architecture was inspired by current reasearch and this thesis did not optimize any hyperparameters or network architecture.

This work was carried out in the facilites of Saab AB, who produces different kinds of radar systems. The real-world data was acquired by an airborne surveillance radar system in a testing facility. The data collection method was thus limited to the functionality specific to that radar system. To use the data for supervised learning it requires labelling. This thesis limits the data collection method to collecting new data and labelling it whilst doing so. As will be shown in Section 4.5.3, data from a Saab ground radar system were also used.

To use the neural network in an airborne radar system it must be able to work under the constraints put upon the computational complexity. Therefore it was considered relevant to propose methods of how to increase computational speed of a neural network.

1.6 Thesis outline

This thesis consist of seven chapters. Chapters 2-3 aim to provide a reader with basic knowledge of radar signal processing and deep learning. Chapter 4 describes how data was collected and labelled, and how the neural network was trained and evaluated. In Chapter 5 the results from the data collection and network evaluation is presented, along with a brief analysis of why they ended up the way they did. In the remaining chapters, a more elaborate analysis of the results is carried out to find weaknesses in the study and draw conclusions. Ethical implications of this work are also discussed along with some ideas for future studies.

2
Airborne radar

This chapter summarizes some fundamental concepts of airborne radar systems relevant to this work. The goal is to provide the reader with understanding of what data could be used for machine learning and what patterns could be expected to learn. For further reading, suitable chapters from [1] are referred to in each section.

2.1 Fundamentals of a radar system

A radar is a sensor that emits electromagnetic energy and measures returned electromagnetic energy, and a common application is target detection. The range of which a target can be detected depends on several factors such as transmission signal power, antenna size, signal wavelength, target aspect angle etc. The received signal is heavily distorted by noise, not only the electrical noise in the radar itself but also in the form of returns from unwanted objects. The unwanted return is called *clutter*. The delay between signal transmission and receiving can be measured to determine the range of the target. If the round trip time is t, the target distance r is then

$$r = \frac{1}{2}tc \tag{2.1}$$

where c is the speed of light. The factor $\frac{1}{2}$ due to the fact that the signal must travel two ways; one trip to the target and then from the target back to the receiver.
If the transmitted signal is reflected from a target that is moving in radial relation to the receiver, the returned signal is also affected by the doppler effect described as

$$f_d = \frac{-2\dot{r}}{\lambda} \tag{2.2}$$

where λ is the wavelength of the transmitted pulse, and $\dot{r}$ is the radial velocity of the target relative to the receiver. For example, for an X-band radar, the wavelength is around 0.03 m, and if the pulse is reflected on a target moving $300\,m/s$ towards the receiver, the Doppler shift will be around $\frac{-(-600)}{0.03} = 20$ kHz. Therefore the ability to measure the doppler shift of the received signal allows for the radial relative velocity of the target to be determined. For more detailed explanation, see [1, Ch. 1, Ch. 15, Ch. 26]. In this thesis, the *aspect angle* is defined as the horizontal target velocity vector relative the radar position, thus a target that moves directly towards the radar has an aspect angle of 0 degrees and a target that moves away from the radar has 180 degrees.

2.2 Pulse-doppler radar

The following section describes the inner workings of a pulse-doppler radar system which can be found in [1, Ch. 2-3].

A pulse doppler radar (PDR) emits pulses of electromagnetic energy and is able to use the doppler shift in the returned signal to determine the velocity of a target. Due to the pulsed operation of a PDR, it is possible to use the same antenna for transmission and receiving, often used in application where physical space is limited, such as in airborne applications. During transmission, no signal can be received, and while receiving there cannot be any transmission. In Figure 2.1, a simple PDR system is shown.

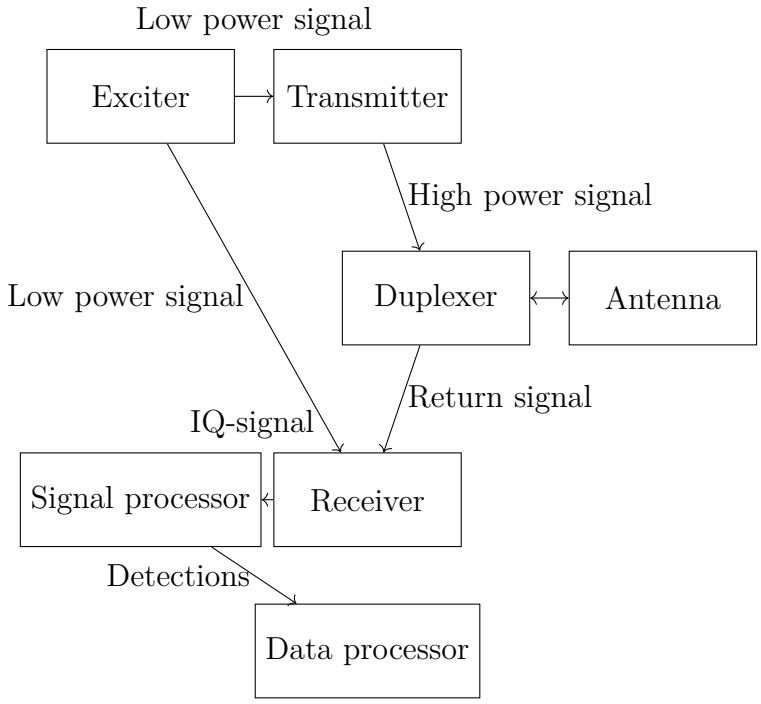

Figure 2.1: Simple illustration of the fundamental elements of a PDR.

The components can be described as

- Exciter: Creates a low power electrical AC signal of desired frequency.
- Transmitter: Amplifies the signal and sends it to the Duplexer.
- Duplexer: Chooses whether to send or receive at a given moment in time.
- Antenna: Converts electrical signal to electromagnetic energy and vice versa.
- Receiver: Demodulates the analog return signal to baseband using the low power Exciter signal. Creates the complex-valued digital IQ-signal.
- Signal processor: Detects targets in IQ-signal.
- Data processor: Filters detections and keeps track of targets.

When the transmitter has sent its pulse, the receiver starts listening for a signal. The received signal is demodulated to baseband (i.e. frequencies around 0), enabling sampling with lower frequencies. The signal is sampled two times, one with the

oscillator signal phase-delayed 90 degrees creating the quadrature- or Q-signal, and one with the non-delayed oscillator signal creating the in-phase- or I-signal. The IQ-signal is then the complex number created by setting the I sample as the real part and the Q sample as the imaginary part of the IQ sample $s = I + Qj$. From each pulse a vector of complex numbers is created as

$$s = \begin{bmatrix} s_1 & s_2 & \cdots & s_M \end{bmatrix} \tag{2.3}$$

where M is the number of samples. The interval between the start of each pulse are denoted as the Pulse Repetition Interval (PRI) and the frequency of which the pulses are sent is the Pulse Repetition Frequency (PRF). The ratio between the length of the pulse and the PRI is the duty cycle. A sequence of pulses which will be used in the signal processing are called a Coherent Pulse Interval (CPI). Figure 2.2 shows a CPI with two pulses and three samples after each pulse. In practice, the PRF ranges from a few hundred hertz to several hundred kilohertz at X-band [1, Ch. 25]

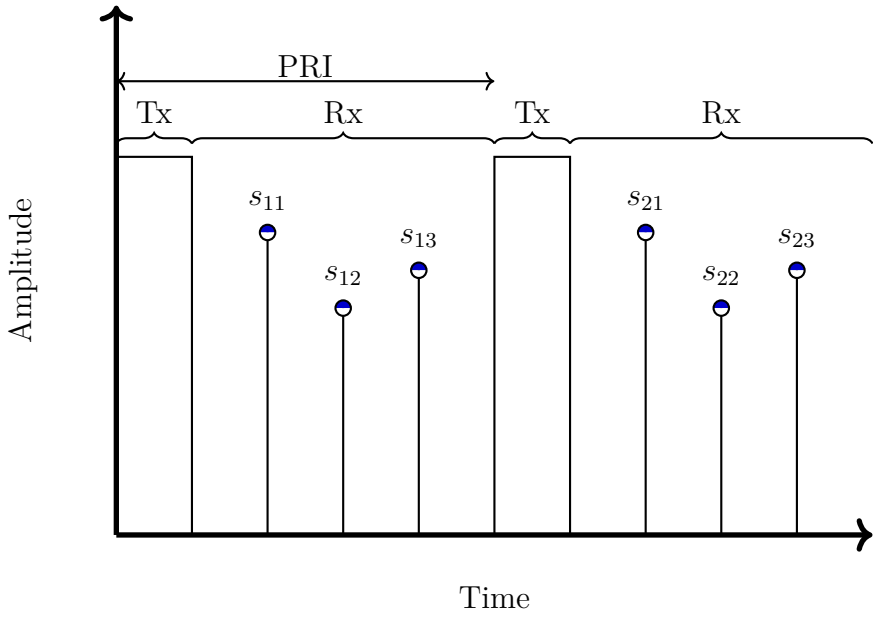

Figure 2.2: The relation between transmitting (Tx) and receiving (Rx) signals in a PDR. The radar sends two pulses and the received signal is sampled three times after each pulse.

In this case, the IQ-signal is gathered in a matrix S as

$$S = \begin{bmatrix} s_{11} & s_{12} & s_{13} \\ s_{21} & s_{22} & s_{23} \end{bmatrix}. \tag{2.4}$$

In the general case of a CPI with N pulses and M samples after each pulse, the IQ-matrix becomes

$$S = \begin{bmatrix} s_{11} & \cdots & s_{1M} \\ \vdots & \ddots & \vdots \\ s_{N1} & \cdots & s_{NM} \end{bmatrix} \in \mathbb{C}^{N \times M} \tag{2.5}$$

Since each sample is taken at trailing time instances, each sample corresponds to a specific range. If the sampling interval is Δt, the i:th sample after each pulse corresponds to a range of $\frac{1}{2}\Delta tc$ meters, from Equation 2.1. Hence the i:th sample is referred to as the i:th *range bin*. To determine how the signal changes at every range, the Discrete Fourier Transform (DFT) is applied to the IQ-matrix over the pulse dimension, creating the range-doppler matrix or RD-map. The RD-map thus contains the frequency spectrum at each range, and each element in the frequency dimension is called a *doppler channel*. In an airborne radar system, the sampling interval corresponds to a range of around 100 m at X-band [3].

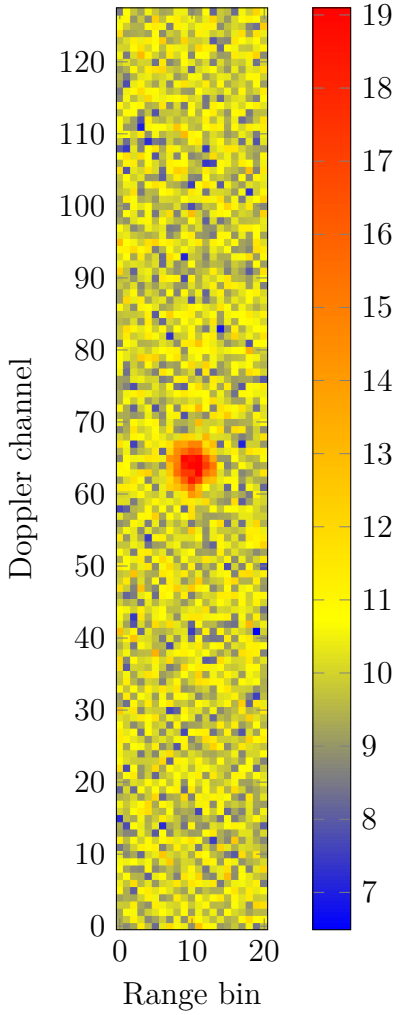

Figure 2.3: Synthetic RD-map created from CPI with 128 pulses and 21 samples after each pulse, resulting in 128 doppler channels and 21 range bins. Each pixel represent the signal strength at a given range and doppler frequency.

Figure 2.3 shows a synthetic RD-map with a target likely present in range bin 10 and doppler channel 64. The RD-map is then used for detecting targets, often using the Constant False Alarm Rate (CFAR) algorithm, and this data representation will be the main form in the rest of this thesis. These matrices come in various shapes and sizes depending on the types of targets to be detected, which will be further

explained in the following sections.

2.2.1 Range- and Doppler ambiguities

Due to the inability to transmit and receive simultaneously in a PDR, there are ambiguities in range and velocity depending on the PRF [1, Ch.12, Ch.25]. Thus it is sometimes impossible to associate a sample to a specific pulse - the returned echo after a given pulse may have come from any previous pulse. Due to aliasing, the doppler frequencies can not always be correctly determined if the sampling rate (i.e the PRF) is too low. The maximum unambiguous range R_u, that is the maximum range that can be unambiguously determined for a given PRF, is

$$R_u = \frac{cT}{2} \tag{2.6}$$

where c is the speed of light and T is the PRI.
The maximum unambiguous doppler f_u for a stationary radar is

$$f_u = \pm\frac{PRF}{2} \tag{2.7}$$

Because of the ambiguities, there are commonly three classes of PRF - low, medium, and high. These are not defined precisely by an exact value, but rather in relation to whether ranges or doppler frequencies are ambiguous or not. At a low PRF, the range is unambigous because the PRF is set such that is unambiguous up to the maximal range of the PDR. High PRF yields unambiguous doppler frequencies while in medium PRF, both range and doppler frequencies are ambiguous. For example, a low PRF is suitable to use at long ranges and a high PRF when the target is moving very fast. Medium PRFs are then suitable for targets that are not at very long ranges and not having very high velocities.

A target with a given range and velocity may appear at different ranges and/or doppler frequencies in the RD-map, due to the ambiguities. To find the true range and velocity, the PDR must use several PRFs to *resolve* these parameters. This means that every CPI may differ in number of pulses and samples between the pulses.

2.2.2 Target tracking and radar modes

Target tracking is the process of estimating the position, velocity and acceleration of a target. Depending on what targets the pilot wants to detect and track over time, different *modes* are used. Two commonly used modes are the Single Target Track (STT) and Track While Scan (TWS). The STT mode is used when accurate estimates of a single target is needed, while the TWS mode is a combination of tracking several targets while searching for new ones. Thus the STT mode is suitable when only one target is of interest while TWS is suitable for more general operation [1, Ch.29]. Both mentioned modes often use medium PRF during operation.

In the TWS mode, tracking is done by filtering the detections done by the signal processing, and determining whether the current detections can be assigned to previously tracked targets, if they are false detections or if they are new, previously untracked targets. Since the PDR depends on using multiple PRFs to accurately resolve the target parameters, the RD-maps might have several different shapes even in the same mode.

2.2.3 Nonstationary vs stationary radar systems

The PDR faces numerous challenges in detecting targets in the noisy radar return. A common issue is the presence of clutter, which can be derived from precipitation or the ground. The ground clutter for a stationary PDR is easier to handle than that of a moving PDR - for a stationary PDR, the ground clutter will have doppler frequences around zero, since the PDR does not move in relation to the ground.
For airborne, or any moving PDR, the ground clutter will be dependent on the PDR velocity and transmission angle, requiring more elaborate filtering techniques to compensate for ground clutter. Airborne PDRs will also be greater penalized by constraints on weight, power consumption and cooling in comparison to ground based PDRs.

Thus, there is a difference in how clutter will appear in the range doppler maps depending on whether the radar is moving in relation to the ground or not.

2.2.4 Micro doppler

Micro doppler is the doppler effect in a target echo not directly associated with the translational velocity of a target. Micro doppler arises from rotating, vibrating components, or any moving part of the vehicle that is seen by the radar [10], and it can sometimes be seen by the naked eye in an RD-map. For example, the rotation of the propellers of an aircraft could produce micro doppler effects. Thus the micro doppler effect can provide additional target information rather than just range and velocity.

For targets as helicopters, the micro doppler effect is likely to be distinguishable in a radar echo regardless of aspect angle, due to its 360 degree visible rotor blades. Targets such as boats have likely very little micro doppler return due to the lack of visible rotating parts, while other forms of aircraft emit detectable micro doppler signal for some aspect angles - such as a jet aircraft that flies towards the PDR having a near zero aspect angle, since its turbines can be seen.

3
Deep learning

This chapter presents theory behind deep learning and neural networks relevant to this thesis, limited to supervised learning. For in-depth explanations, see [9].

AI is an active research field, and Goodfellow et. al. [9, p. 1] considers every computer program as some kind of AI. Early AI solved problems that may be expressed as a set of mathematical rules, while the emerging AI technologies such as machine learning aims to solve problems that are not as easy to express in a mathematical context [9, p. 1].

3.1 Machine learning and neural networks

Machine learning is the property of an algorithm to develop itself by feeding it raw data [9, p. 2-3]. Deep learning is a subset of machine learning which dates back to the 1940s, and has had several names ever since, gaining popularity in 2006 [9, p. 12-13]. A deep learning algorithm is a machine learning algorithm that learns complex representations as a series of simpler representations [9, p. 5]. As mentioned, one of the many names the deep learning algorithms have gone by has been artificial neural networks, due to thier resemblance to neurons in biological brains. A deep learning algorithm is often referred to as a *neural network* or just *network*, sometimes also *deep learning model* or just *model*.

The most common type of ANNs are the feedforward neural networks, which is approximating some true function $y = f^*(x)$ as the network function $\hat{y} = f(x;\theta)$ where x is the input data and θ is the network parameters/weights [9, p. 164].

In the simplest case, the input is a vector $x = [x_1, x_2, ..., x_n]^T$ and the weights θ is the set containing both the multiplication weights W and bias weights b. Figure 3.1 shows a neural network with one layer containing one neuron/unit. This function maps a vector $x \in \mathbb{R}^n$ to a scalar $f(x) = \hat{y} \in \mathbb{R}^1$. A network with m output neurons maps the input to $\mathbb{R}^m$ instead.

A neural network more commonly consists of several layers, which in themselves can be seen as simple mathematical functions. Each layer essentially computes the matrix multiplication between the input vector and its multiplication weights, then adds the bias weights. This sum is then run through some differentiable function σ called the *activation function*, such as the *rectified linear unit* (ReLU).

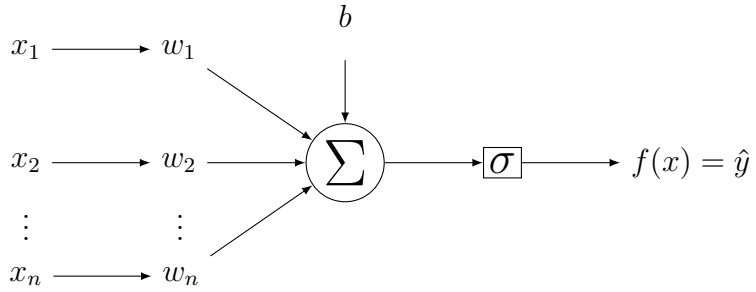

Figure 3.1: A simple neural network consisting of one layer and one neuron. The input x is a vector with n elements, and there are n multiplication weights (w) and one bias weight (b). σ is the activation function.

If the input to the first layer $f^{(1)}$ is x, then the output of that layer can be expressed as

$$f^{(1)}(x) = a^{(1)} = \sigma(W^{(1)}x + b^{(1)}) \tag{3.1}$$

where $a^{(1)}$ is called the layer activation, $W^{(1)}$ is a matrix containing the multiplication weights and $b^{(1)}$ contains the bias weights for the first layer.

The activation from one layer acts as the input for the next. For instance, a neural network f with three layers can be described as

$$f(x;\theta) = f^{(3)}(f^{(2)}(f^{(1)}(x))) \tag{3.2}$$

when omitting the network parameters $\theta^{(i)}$ for $i = 1, 2, 3$.

In general, the output at layer i can be expressed in terms of layer $i-1$ as

$$f^{(i)}(a^{(i-1)}) = a^{(i)} = \sigma(W^{(i)}a^{(i-1)} + b^{(i)}). \tag{3.3}$$

3.1.1 Convolutional neural networks

Convolutional Neural Networks (CNNs) are a special case of feedforward networks which employ a convolution operation, though not identically defined as the convolution found in e.g. signal processing [9, Ch. 9.1] as the filter is not reversed. CNNs were popularized in 2012 when a CNN outperformed current state-of-the-art algorithms in an image recognition benchmark test [11]. The main advantages of using CNNs are that the CNN needs fewer parameters and is translationally invariant in the input [9, Ch. 9.2]. Translational invariance enables the network to detect features that look the same but are located in different parts of the input. The need for fewer parameters is accomplished by reusing weights for several parts of the input, while this parameter reusage also results in the translational invariance.

Goodfellow *et al.* [9, p. 326], defines CNNs as "... simply neural networks that use convolution in place of a general matrix multiplication in one of their layers." For

a convolutional layer, the matrix $W^{(i)}$ of (3.3) is a sparse matrix that performs the convolutional operation when multiplied by the activation from the previous layer. The convolution operation is commonly performed in one or two dimensions, and in the 2D case, the output of the convolutional layer is also two dimensional.

A convolutional layer consists of several filters or *kernels* (often a number 2^k for some integer k), which in the 2D case are same-size matrices of size $n \times m$. The activation from the previous layer is assumed to be a matrix, and each kernel is then multiplied in a sliding window manner over the activation output from the previous layer, creating a new matrix. Since this is done once for each kernel, the output from the convolutional layer is a three dimensional array.

Shaping the output from the layer can be done by choosing *padding* and *stride*. Padding is the process of concatenating zeros around the input thus enlarging the output array, which can be done such that the output array has similar dimensions to the input (referred to as having *same* padding). Stride is how the sliding window operation is carried out - a stride of 2 x 2 means that the kernel moves two rows or two columns for each multiplication. For more in-depth explanation, see [9, Ch. 9].

3.1.2 Recurrent neural networks

An RNN differs from a feedforward network by its ability to feedback information from one sample to another. This is because the RNN contains additional variables, creating a hidden state of the RNN which at timestep t is

$$h_t = f(h_{t-1}, x_t). \tag{3.4}$$

There exist several examples of RNNs, which can define their hidden state in other ways, but the main idea is to have variables that enable retaining information between timesteps.

A popular RNN is the Long Short-Term Memory (LSTM), which has become successful because of its capability to retain information for many timesteps. The LSTM is described further in [9, Ch. 10.10], while [9, Ch. 10] explains RNNs in general.

3.1.3 Training and development of deep learning algorithms

Developing a deep learning algorithm/neural network is a data-driven process thus highly dependent on the data that is used in the process. In supervised learning, given a data set

$$\mathbb{D} = \{\mathbf{x}, \mathbf{y}\} \tag{3.5}$$

where $\mathbf{x}$ is the set of input values and $\mathbf{y}$ is the set of output or *target* values, each element in the subsets $\mathbf{x}, \mathbf{y}$ form a matching pair (x_i, y_i). These sets can be described

as

$$\mathbf{x} = \{x_1, .., x_N\} \tag{3.6}$$
$$\mathbf{y} = \{y_1, .., y_N\} \tag{3.7}$$

where N is the number of samples. During training, the network learns the mapping

$$f : \mathbf{x} \rightarrow \mathbf{y} \tag{3.8}$$

by observing the data pairs. Each pair $\{x^{(i)}, y^{(i)}\}$ can be modelled as random samples from the training data distribution $\hat{p}$, that is

$$(x^{(i)}, y^{(i)}) \sim \hat{p} \tag{3.9}$$

which in the best case is similar or identical to the true distribution p

Since the dataset $\mathbf{x}, \mathbf{y}$ is already known, running the algorithm on $\mathbf{x}$ is not very interesting - the true answers $\mathbf{y}$ are already known. What is interesting is how the algorithm performs on data that has not been observed, since this is the most probable use of the network. During algorithm development, the available dataset is commonly split into three parts - the training, validation, and test sets.

The training set is the data that the network uses for modifying its weights θ to fit the function $\hat{y} = f(x; \theta)$ to the true function $y = f^*(x)$. The validation set is indirectly used in training by measuring the network performance on this data during training. When the network has been trained, the test set performance is measured to simulate completely unseen data, as if the algorithm was run in the intended application. In practice, these subsets are often created by choosing subset sizes and then randomly sampling the dataset until the subset sizes are attained. The relation in size of these sets will be further denoted as "70:10:20 split" for a dataset split into parts of 70 % training, 10% validation, and 20% testing. However, this way of splitting the data for verifying the results assume that there is no bias in the dataset, as will be further explained in Section 3.3.

The performance of these sets provide a basis for analysis. A network that does not improve its performance on the training set may indicate that the network architecture is unsuitable or that the data does not have sufficiently rich features to learn. Better performance on the training data than on test data may indicate that the network has been trained too much, often referred to as *overfitting* [9, Ch. 5.2]. Overfitting indicates that the network fails to generalize - it does not perform as well on data that is has not seen. Poor generalization may also be due to dataset bias, which will be further explained in Section 3.3. If the network fails to perform well on the training set, it is considered *underfitted*.

Any machine learning algorithm is evaluated by some performance criterion, such as accuracy in a classification task. The accuracy is determined by dividing the number of correct guesses by the total number of samples [9, ch.5]. Finding the

optimal set of weights θ is thus an optimization problem aiming to maximize performance. Solving this particular optimization problem is often not possible and hence another objective function is used in optimization, hoping that it will improve the performance measure [9, ch. 8].

The objective function $J^*(\cdot)$ is the expected value of a loss function over the data-generating distribution p, for some loss function $L(\cdot)$, as

$$J^*(\theta) = \mathbb{E}_{(x,y)\sim p}[L(f(x;\theta), y)]. \tag{3.10}$$

Since the true distribution p is not known, the samples are instead following the distribution $\hat{p}$. The objective function is then approximated as follows

$$J^*(\theta) \approx \mathbb{E}_{(x,y)\sim \hat{p}}[L(f(x;\theta), y)] \tag{3.11}$$

where $\hat{p}$ is the training set distribution. Because the training set is finite, the expected value is approximated as the sample mean

$$J^*(\theta) \approx J(\theta) = \frac{1}{N}\sum_{i=1}^{N} L(f(x;\theta), y) \tag{3.12}$$

where N is the number of samples. This function is then optimized by randomly initializing the weights θ and updating them by running some gradient descent algorithm. The gradient g is calculated by taking n samples $\mathbf{x} = \{x^{(1)}, .., x^{(n)}\}$, $\mathbf{y} = \{y^{(1)}, .., y^{(n)}\}$ from the training set and approximated as

$$\hat{g} = \frac{1}{n}\nabla_\theta \sum_i L(f(x^{(i)};\theta), y^{(i)}). \tag{3.13}$$

Thus the gradient estimate is the mean gradient of the loss over all the samples in the *minibatch*. The size of the minibatch is often referred to as *batch-size*, and is commonly a power of two, depending on how many samples that fit into memory, most likely 32, 64, 128, or 256.

As mentioned, the weights are updated using a gradient descend method, which uses the estimated gradient from each minibatch to update the weights as

$$\theta_{k+1} = \theta_k - \lambda\hat{g} \tag{3.14}$$

where k is the optimization step number and λ is the step size or *learning rate*. This is one of the simplest form of optimization, in practice more advanced variants are used, such as RMSprop or Adam [9, Ch.8.5].

When the network has been trained with all the data in the training set one time, one *epoch* has passed. If the training sets are big enough, one epoch might suffice, but often several epochs are used for training. If the total size of the training set is N and the batch size is n, then the total number of optimization steps taken during one epoch is the result of the integer division $\frac{N}{n}$.

3.2 Neural network pruning and acceleration

The recent success of neural networks has been accompanied by large models, requiring vast amounts of computing resources for training and inference [12, 13, 14, 15]. These large models are often over-parameterized [16], which makes them susceptible to overfitting. [15]

Reducing the computational complexity of the networks is not only important to reduce inference time - in embedded or other resource constrained environments the power consumption is also a factor that must be considered. But what affects power consumption? Han *et al.* [15] suggests that arithmetic computations are rather cheap in comparison to memory accesses. SRAM memory access is in the same order of magnitude in energy consumption as a 32-bit floating point addition, while DRAM access is three orders of magnitude more energy consuming than a 32-bit floating point addition. Hence limiting memory access is a valuable tool to enable deep learning in resource constrained environments.

Pruning is a method employed to address the issues derived from having extremely many parameters in a model, a field where one of the early pioneers were LeCun *et al.* [17] in 1989, followed by Hassibi *et al.* [18]. The method aims to find elements in a neural network, such as weights, layers, and filters that contribute the most to the prediction capacity, the highest *saliency*. When the saliency has been determined, the elements that have low saliency are removed. As weights are removed (i.e. set to zero) and the weight matrices become sparser, the network can be compressed to require less storage space, such as up to one 63:rd of the ordinary network size [19].

This type of pruning is called *unstructured pruning* and it does not yield immediate performance or memory benefits if not specialized software or hardware is used, since its sparsities cannot be exploited [20, 12]. Yet unstructured pruning may be used as a tool to combat overfitting [21], and can often be used to remove a decent amount of weights without losing performance, sometimes even gaining performance, on large networks [15, 22].

Structured pruning on the other hand is the process of removing higher-order elements such as layers or filters, which gives faster inference and lower memory usage even in standard libraries [14, 23, 24].

A problem in all pruning methods is that the saliency is not some quantity that can be observed - it is merely a quantity or criterion determined by the developer, hence the holy grail of pruning research is to find the optimal saliency measure. The magnitude of the weights were a common saliency criterion prior to 1989 [17], regaining popularity in 2015 as an scalable and efficient criterion [15, 12]. By this criterion, low magnitude weights are considered to have low importance, a belief which is questioned in [17, 23].

Thus, a significant difference between pruning methods is the salience criterion, but

how elements are removed may differ aswell. A typical pruning method consists of training a model (or using a pretrained), removing weights, and fine tuning to regain performance [16]. This type of pruning is used in [15], while [22, 12, 23] instead uses an iterative approach, mixing element removal with fine tuning.

These methods require training a model or having access to a pretrained model, which states that someone has to make the effort to train the model before pruning. Zhu and Gupta [12] conclude that pruned models (large and sparse) outperform small and dense models for a given memory size. Frankle and Carbin [25] proposed the *Lottery ticket hypothesis*, which states that a randomly initialized feedforward network before training contains subnetworks that can achieve comparable performance to the whole network. These subnetworks are referred to as *winning tickets* and can be found by unstructured pruning. These winning tickets are further investigated by Zhou *et al.* [26] who suggested that the only important components of the winning tickets are the sign of the weights from the beginning, thus it is possible create networks that do not need training to have at least some performance.

Liu *et al.* [16] showed that for structured pruning, it is not necessarily true that pruned models outperform smaller nonpruned models, but that the same performance can be achieved by directly training a small model with the architecture found by structured pruning. They argued that the structured pruning methods may be more suitable in finding a good network architecture instead of actually using the same weights as the pruned network. At the same time they achieve contradictory results from those in [25], stating that the winning tickets does not achieve better performance than random initialization when using large learning rates.

There are also other methods for optimizing neural networks that such as quantization, low-rank factorization, knowledge distillation and compact convolutional filters [13], but these methods will not be further explained in this thesis. Even though a network is *possible* to implement in a real-world application, it is necessary to investigate whether it *should* be implemented. The developer must ensure that the network does what it is expected to do when implemented in the target environment, making sure that there was no bias in the data used for training.

3.3 Dataset bias

As the performance of any machine learning algorithm is the product of the data that was used in training, the dataset essentially determines the performance of the algorithm. A core machine learning problem is gathering the proper data to feed the algorithm. But what is "proper" data? The whole point of the algorithm is to have good performance on data it has not been trained on [9, Ch. 5.2], and to accomplish this, it is necessary for the training data to be sufficiently similar to the data that it will encounter in the intended application. Thus proper training data is data that resembles the data generated in the application well enough - an *unbiased* dataset.

It is often impossible to sample all the possible data that the algorithm might encounter in its application, thus the algorithm must learn to generalize - learn patterns and features in the training data that will enable it to have good performance on unseen data. A datapoint generated in the application can be seen as a sample from a random variable that follows an unknown distribution p, while the training data can be considered samples from another random variable distributed as the unknown distribution $\hat{p}$.

A proper training set is such that $\hat{p}$ is similar, or in best case identical to p. Thus the training samples must be similar to the samples generated by p, but also sufficiently many to capture all variations of p. The training set is biased if $\hat{p}$ is not identical to p, which in practice is always the case, the question is if the training set bias is sufficiently small for the algorithm to perform well in the application. As previously mentioned, the dataset is usually randomly split into training, validation and testing, which is a form of *cross-validation* (CV) [9, Ch. 5.3]. However, measuring performance only on these sets may be misleading if the dataset is biased.

Bias may be hard to detect and can lead to entirely wrong conclusions as a problem may appear easier than it actually is. It has been shown that many popular benchmark datasets are biased [27].

An example of a biased training set is when developing an algorithm to distinguish cats and dogs in images. If all the images with a dog for some reason also contained bicycles, and all the cat images also contained cars, what would the network learn? To distinguish dogs from cats or bicycles from cars? Due to this erroneous data collection procedure, it is not certain. In this case $\hat{p}$ is a poor representation of p since certainly dogs and cats occur in images without vehicles. In this case, the bias could have been detected by manual inspection of the images or by selecting the test set from another data source (hopefully not contaminated by vehicles) and seeing the performance degrade significantly.

Yet bias is harder to detect when data is not as intuitive to humans as images, or when the bias is less distinct. It can be thought of as any characteristic in the training data that correlates samples to each other but is not a useful feature to base a prediction on. In the previously mentioned cat and dog example, bicycles and cars are features/characteristics present in the images but to humans it is obvious that the vehicles have nothing to do with cats and dogs, yet those are characteristics correlating the samples, a pattern that the algorithm will find.

In data representations less intuitive to humans it is hard, or even impossible to find bias by manual inspection. If there exists some characteristic that seems to separate classes of datapoints, how would a human be able to determine whether this characteristic is a useful feature or bias? This would require that the person inspecting the data has domain specific knowledge to subjectively evaluate what features in the data that could be useful and what could be bias.

When bias is hard to detect by manual labour, CV may be used.

3.3.1 Cross-validation

CV is the process of splitting the dataset into two partitions, using one partition for developing the predictive model and the other for evaluation [28]. Selecting these partitions can be done *controlled* or *uncontrolled*, and Stone [28], suggests that an controlled CV is when the samples in the dataset are chosen randomly. As earlier mentioned, performing a common training, validation and test set split randomly is not only an example of CV, but also a controlled CV. From [28], *uncontrolled* CV can be interpreted as when the split is based on some known parameter, such as time.

A *k-fold CV* is the process of splitting the dataset into k partitions and for k iterations selecting one partition as the test set and the other $k-1$ partitions for training [9, Ch. 5.3.1]. This is done such that each partition has been used for testing one time, and the performance is averaged over all iterations. In Figure 3.2, a partitioning of the dataset from a 4-fold CV can be seen.

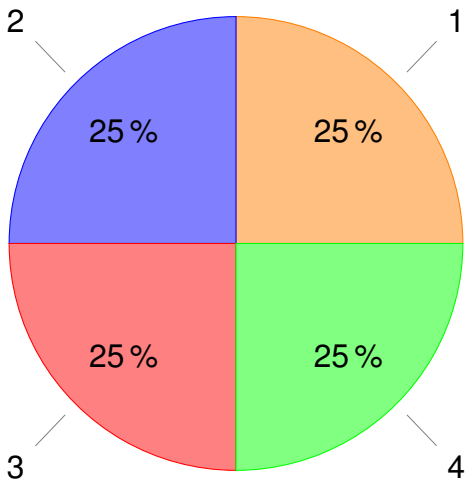

Figure 3.2: Example of partitions in a 4-fold CV scheme. In the first iteration, partition 1 is chosen as test set and the network is trained on partitions 2,3, and 4. This is done until all partitions have been used as the test set one time, and the performance is averaged.

If the performance on the partitions are significantly different, the dataset may contain bias. However, by performing a controlled CV, it is not possible to pinpoint the source of bias, since the partitions are randomly chosen. Intuitively, if the partitions are not random but instead known, such as in an uncontrolled CV, it is expected that more information of the source of bias would be known. By having access to further information of the dataset, as *metadata*, each known parameter in the metadata could potentially be used in an uncontrolled CV. Using metadata for partitioning the dataset, will be further referred to as *metadata CV* (MCV). MCV and how it is expected to be valuable in finding dataset bias will be more thoroughly

explained in Section 4.3.

Another way of detecting bias is to investigate what features the network has learned. If the network has learned useful features to base its prediction on, there was no or little bias in the dataset. Dataset bias and unuseful features learned are tightly coupled since bias increases the likelihood of learning unuseful features, and concluding that the network has learned unuseful features indicates dataset bias. But a problem arises - how can a human determine whether the network has learned useful features or not? The human must have domain specific knowledge in order to understand what useful features *could* be found in the data, making a subjective assessment whether the features are useful or not. But there must also exist some method to explain the actions and decisions to humans, which is a problem since neural networks are traditionally black boxes. The solution is called *Explainable AI*.

3.4 Explainable AI

Explainable AI is the gathering of methods which analyze the decision making of the machine learning algorithms in an apprehensible way to humans. The inability to understand why an algorithm took certain decisions is limiting the usefulness of the algorithms [29], simply because humans are unlikely to trust and implement an algorithm they do not understand. Since different people need different information in order to understand or trust an algorithm or model, Arrieta *et al.* [30] defines explainability in the following way: "Given a certain audience, explainability refers to the details and reasons a model gives to make its functioning clear or easy to understand". Thus whether a model is explainable or easy to understand is completely dependent on what audience its explanations shall be presented to.

Several methods exists to give human understandable explanations on why a deep learning model has made a certain decision, such as Layer-wise Relevance Propagation (LRP) [31], intermediate activation visualization [32], and several more which can be found in [30].

3.4.1 Occlusion

Understanding a CNN can be done using a method proposed by [32, Ch. 4.2]. The idea is simple - by removing the information in a set of pixels (patch) in an image, such as setting those values to zero, the output of the network without that information is measured. The patch is then slid over the entire image, and by storing the output of the network from each occluded position, a new image can be created. Thus the output of this process can be a new image where each pixel represents the most probable class, or the probability of a given class when that portion of the image is occluded.

From this method the developer can "see" what parts of the input image had large importance for the prediction - if the predicted class or the prediction probability changes much as a certain area is occluded, that area is assumed to have some

relevance to the prediction. However, due to the nonlinear behaviour of deep neural networks, this method would not be able to capture when several distinct parts of the image together constitute the reason for the prediction. It is still useful and easy to apply since it does not need any addditional information of the network, in contrary to other methods.

3.5 Deep learning in radar signal processing

Classification and object detection using deep learning is an active research field, using images as input data. The task can be stated as labelling a whole image[11], labelling every pixel [33] or finding the position and extension of objects [34, 35, 36]. Object detection is essentially a classification problem with only two classes - object or no object. Performing detection in a radar signal processing is thus solving a classification problem, and while deep learning has been successful in classification problems in image recognition, representing radar data as images enables using methods from the field of image analysis.

Wang *et al.* [6] created a neural network to perform radar target detection. This algorithm consisted of running a binary classification network in a sliding window manner over the RD-map. They conclude that their neural network algorithm has better performance than a traditional detector under some conditions but since they only used synthetic data, further research on the algorithm would require real-world data to evaluate its performance. This approach is also rather inefficient, since the network must be evaluated many times per image. Brodeski *et al.* [37] proposed a detection method based on the Faster R-CNN algorithm [35], allegedly being able to run a sample in 20 ms, yet without specifying computational hardware. They used data from a calibration process, thus the data was partially real - they did not include data from any operational scenario. However, their algorithm also determined target elevation- and azimuth angles.

Earlier research in classification of radar targets often uses high resolution range profiles (HRRP) [38, 39, 40, 41] or micro doppler effects [8, 42, 43]. Other approaches include target flying patterns [4] and raw time-domain data [44]. Since RD-maps are essentially images, an intuitive method for target classification is utilizing CNNs, but as radar data often occurs as a stream of data over time, there are often gains in utilizing RNNs [38, 39, 44, 5, 7].

Altmann *et al.* [5] used RD-maps in distinguishing six different states in a continuous cooking process. When utilizing single RD-maps for classification, they achieved an accuracy of 60 %, which was improved to 99 % when adding temporal information by RNN network structures. Kouba [38] and Jithesh *et al.* [39] used HRRP with RNN and achieved good performance on classification tasks, although solely synthetic data was used.

4
Method

This chapter describes the implementation process of building a protoype of a function in an airborne radar system, using deep learning.

The following sections describe the process of radar data acquisition and target association, the process of finding a suitable problem and creating a neural network. Furthermore a custom method for detecting dataset bias is presented and finally several test cases used to evaluate performance is described.

4.1 Creating a dataset with an airborne radar system

This section starts from the very beginning of the design process with collecting data using an airborne radar system, labelling it and processing it into labelled range-doppler maps. The method is a combination of a manual process of data collection and an automatic approach of labelling samples of interest.

4.1.1 Raw radar data acqusition

The airborne radar system used for collection of real-world radar data was an airborne surveillance radar. Recording of data was done in a testing facility, meaning that the radar system was stationary on the ground. As earlier mentioned, this will make the clutter appear different in the RD-maps from when the radar is moving.

To record data, the testing facility had to be operated manually, and the operator had to choose which targets to collect data from and what radar modes to use. The concept of ”recording” data is here referred to as operating the radar ”live” and save the received information, along with numerous target parameters from the tracking system. During radar operation, the target detection and tracking system found airborne targets flying over the Gothenburg region. During the search for a suitable target no data was saved. When a target was located the recording started. The information contained in a recording was the raw IQ data associated with tracking system information, such as position, altitude and heading of the targets. Each target found by the radar system was assigned an identification number by the tracker. As will be shown in Section 5.1, the targets mostly consisted of commercial jet aircraft.

To select targets and label them, data from an air traffic control (ATC) system was collected simultaneously as the recordings. This data consisted of several parameters such as target position, altitude, source and destination airport, callsign, aircraft type (ICAO Type Designator, a four letter string describing the aircraft type) and a unique aircraft registration number. The operator chose targets from the ATC data that were believed to be within detection range of the radar and on adequate height.

4.1.2 Automatic labelling of data

A data driven approach often requires large amounts of data and following method is a strategy to automatically label data after each recording, since supervised learning require input-output pairs. Labelling each recording by hand would make the data-driven development procedure infeasible since then the development time would be too lengthy.

Commercial aircrafts send data with their transponder to an ATC service when passing through different airspaces. The data from the transponder contain information about the aircraft, such as position, altitude, heading and etc. The data from the transponder is similar to the estimates prodced by the tracker system. This work thus consider the transponder data as the 'true' one and the ATC information can thus be utilized as labels. However, in reality the transponder data is also an estimation. Yet, these different sources of data are not possible to combine without some adjustments. Different sample time sets them apart, and time of continuous tracking. In addition, an ATC service keep track of all aircraft in the airspace, hence the data will contain a huge amount of aircrafts unlike the data from the tracker system. Therefore interpolation and search in the ATC data was necessary to determine what aircraft in the ATC data matched the target chosen during radar operation.

Scripting the labeling process is structured from the way the data is recorded by the test facility. The recordings are folders with raw IQ data stored on a file server. When the operator decides which target to aim for and record, it becomes one folder. Recording multiple targets therefore generates many folders of IQ data which have a corresponding data package with information from the tracker system. If one package only contains information from using STT mode all samples would contain the target, this is however unusual since the TWS and STT modes are used interchangeably. Since the majority of information fram a scan mode is irrelevant to this work a stategy to sort it out is necessary.

Since each target is recorded in one package it is enough to sort for the longest prioritized track in each package. That would provide an identifier unique to the target assigned by the tracker, which makes it possible extract the prioritezed targets coordinates, altitude and time fromt the metadata. With the identification number from the tracker it was possible to extract the chosen target's coordinates, altitude and time from the tracker data. The next step was to make sure all data uses the

same units as in the ATC data, which utilizes UTC-time in seconds from midnight, altitude in meters and coordinates in the WGS84 coordinate system.

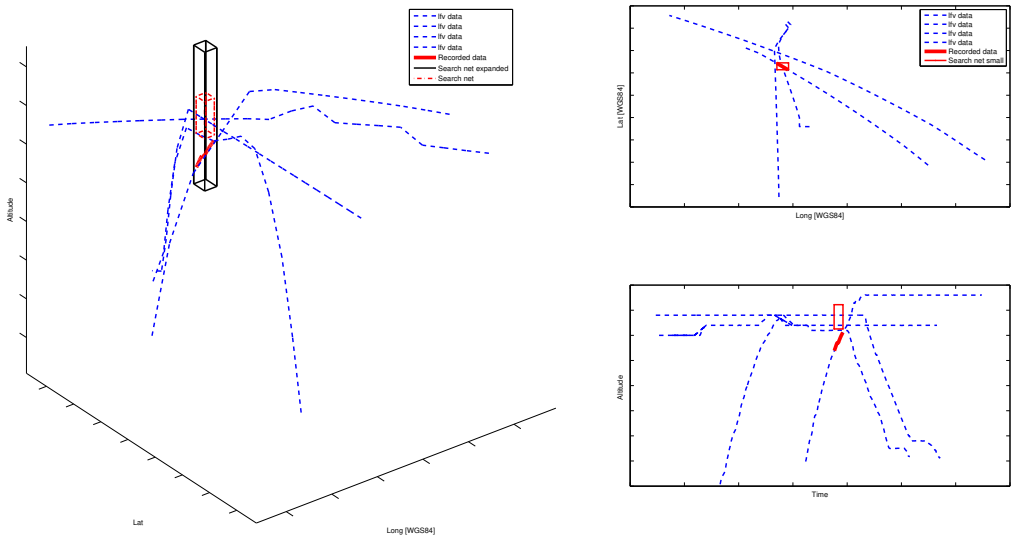

Figure 4.1: How to extract label from ATC data and apply to the collected radar data. The dotted lines are different trajectories of aerial targets from recorded ATC data. The thick line is the interval for when the radar system recorded the aerial target. The squares visualize how the search program identifies the correct trajectory and matches it with the radar system's recording.

As mentioned, the data from the tracker system and the ATC data are set apart by different sample intervals and length of the intervals for continuous tracking, even though the units now are the same. As shown in figure 4.1, only searching using two parameters could yield an ambiguous result. Meaning that two different airplanes could exists in the choosen search window and the correct label could not be determined. By creating a four dimensional search window with data from the tracker system (latitude, longitude, time and altitude), the corresponding aircraft in the ATC data was found. It was necessary to use atleast coordiantes and time. The search window created by the tracker information is most likely to small, and it was necessary to expand it to find the target. The estimates from the tracker and corresponding offsets used were:

- Latitude WGS84 [deg] $\pm$ 0.5 [deg]
- Longitude WGS84 [deg] $\pm$ 0.5 [deg]
- Time UTC [sec] $\pm$ 3 [sec]
- Altitude [m] $\pm$ 2000 [m]

It could however still return an ambiguous result. Then it would be necessary to split the intervall of coordinates and time into smaller intervalls and create multiple smaller search windows. Chosing the most frequently occuring aircraft in all of the smaller search windows provide a final result. For this work it was however only

necessary to search using the full interval. If the search process had successfully matched a tracker identification number to an aircraft in the ATC data, then it was possible to associate each IQ matrix with any parameter available in the ATC data.

Finally one last category was added to the labeld dataset, which was 'no-target'. In those datapoints there are no detected targets and they only contained background, such as clutter.

4.2 Commercial aircraft classification using deep learning and RD-maps

Before training a neural network, the IQ matrices were processed into RD-maps. A raw radar data frame consisted of the complex IQ-matrix of size N rows and M columns, where N is the number of pulses and M is the number of samples after each pulse.

Each pulse was windowed with a Hann window to reduce sidelobes in the signal introduced by sampling. The fast fourier transform (FFT) was applied over the range dimension (which become the x-axis in the RD-map) to transform the matrix into the frequency domain, creating the RD-map. Since the RD-map consisted of complex numbers, only the magnitude of the complex numbers were considered.

The number of pulses and number of samples after pulses varied between CPIs, making it infeasible to run all through a network requiring same-size input for all samples. Hence the FFT operation was zero-padded (or only the first 128 pulses in the CPI were chosen) such that the RD-map always had 128 doppler channels, while only 21 range bins were considered, 10 bins on each side of the target. Thus the samples all had 128 rows and 21 columns, suitable for a neural network. Thus the target was expected to be centered in the range bins. There could exist multiple detections in each RD-map, and the target of interest was chosen as the detection with the greatest amplitude.

4.2.1 Choice of deep learning task and network structure

Because the radar data could be represented as images and the ICAO Type Designator was available in the data, a classification task was chosen. This problem formulation is a well studied deep learning task - given an image, determine what class it belongs to. The algorithm aimed to determine aircraft type (as the ICAO Type Designator) using an RD-map as input, learning the targets micro doppler signatures.

The network structure was inspired from that of [5] since they also used micro doppler classification. The architecture is shown in Table 4.1.

Table 4.1: Architecture of neural network. Three convolutional layers were followed by three fully connected layers. The number of classes is denoted as n was dependent on how many classes that were represented in the dataset.

Type	Size	Kernel size	Stride	Padding	Activation
Convolutional	32	3x3	2x2	Same	ReLU
Convolutional	64	3x3	2x2	Same	ReLU
Convolutional	128	3x3	2x2	Same	ReLU
Fully connected	256	-	-	-	ReLU
Fully connected	256	-	-	-	ReLU
Fully connected	n	-	-	-	Softmax

To reduce risk of overfitting and increase training speed, dropout of rate 0.4 and batch normalization were used between all layers.

Each aircraft class was represented as a one-hot encoded vector, and the network was trained with the categorical crossentropy loss, using the Adam optimization algorithm with the default parameters [9, Ch. 8.5.3], for 100 epochs or until the validation loss had not increased for five epochs. Each batch consisted of 64 samples.

For imbalanced datasets, class weighting were used such that the loss of the least frequently occurring class were weighted with one and the i:th class were weighted with $\frac{n_{\text{least}}}{n_i}$ where n_{least} is the number of occurrences of the least occurring class and n_i is the occurrences of class i. For implementation, Keras [45] and Tensorflow [46] was used.

Due to the importance of the data in a data-driven development process, the data collected must be thoroughly understood. CV was used to measure generalization and detect dataset bias. To be able to investigate bias sources, a custom CV scheme was developed, which was derived from the uncontrolled CV previously mentioned in Section 3.3.1.

4.3 Metadata cross-validation

As earlier mentioned, CV schemes such as the k-fold CV cannot be used to find the source of bias, just to indicate whether or not bias exists in the dataset. It is therefore very hard to modify the data collection method to remove the elements that introduces bias, since the source is not known. Therefore a method to detect dataset bias and its source is proposed in this section. By having additional information of the dataset as metadata, the CV could be performed in a way that is expected to help finding what causes the bias, a *metadata CV* (MCV).

The idea is simple - use metadata to partition the samples in the dataset before doing CV, instead of distributing them randomly. Like in other CV methods, differences in performance on these sets may indicate bias, but since it is known what parameters that differs between the partitions, the bias can be pinpointed to some

extent.

MCV is not guaranteed to find the cause of bias, but it indicates that the bias is correlated to the parameter used for partitioning the data. The intuition behind this method is that if the data is partitioned based on a parameter that should not influence the algorithm decision making, but this partitioning actually does influence it, then the partitioning parameter is correlated to the bias source.

For example, assume that a given dataset consists of images of cats and dogs. Also assume that these images had been captured by a few different kinds of cameras. It is not desired that the cameras used for taking the pictures should produce images with such unique characteristics that it is learned by a machine learning algorithm. In this case, the parameter (what camera used for capturing the image) is not desired to affect the images very much. But if it does, the camera used is influencing the algorithm's decision making, or is correlated to something that does. Splitting the dataset into partitions based on which camera took the picture could indicate that the cameras are a source of bias. This bias would have been detected if it was harder to recognize cats taken by a specific camera when not trained on the given camera. As the number of unique cameras approaches infinity, this bias vanishes. Hence, it is expected that having only a few variations of a parameter increases the likelihood that this parameter is bias correlated.

There are however risks with partitioning the dataset. Splitting the dataset may introduce class imbalances, and having classes in the test set not represented in the training set would yield a misrepresentative performance estimate. This could be mitigated by e.g. carefully choosing the partitions or removing samples of classes that are not represented.

Once the partition parameter has been chosen, another CV methods can be used such as the k-fold CV. It is also possible to further partition the data by choosing another parameter to further split on, creating a *nested* MCV scheme. Nested MCV can be performed in an arbitrary number of levels, only limited by the number of metadata parameters available.

The point of using muliple levels is to further investigate what the bias could be, by separating the data using more parameters. For example, choosing targets from an interval of ranges and then splitting that further could indicate how important the range is for the generalization. However, choosing more specific and possibly smaller subset of interest from the dataset leads to a less conclusive results since the sample size of each subset decreases.

4.4 Performance evaluation by cross-validation

Five types of CV was used to evaluate the network performance and will be referred to as simple CV, 10-fold CV, 10-fold MCV, simple MCV, and nested MCV. The

simple CV consisted of partitioning the dataset into a 81:9:10 split. 10-fold CV was performed as a controlled k-fold CV previously described in Section 3.3.1 with 10 partitions. The 10-fold MCV consisted of sorting the dataset by recording time and performing a 10-fold CV. Simple MCV means that the dataset was partitioned on recording occasions and then all samples from one occasion was chosen as test set and the rest of the samples were used for training and validation (10 % validation). The reason for choosing recording occasion as the splitting parameter in the simple MCV is because it was deemed the most accurate way of simulating that the network was used in the real environment, since the network is unlikely being developed the same day as it is implemented.

Nested MCV was done in the following way:

1. **First partitioning**: Choosing a subset of interest from the datasets using the metadata, such as excluding all samples with a given characteristic found in the metadata.
2. **Second partitioning**: Further splitting this subset of data into training, validation and test sets. This was done by either
   (a) simple CV
   (b) simple MCV
   (c) both simple CV and simple MCV.

For any CV scheme, depending on how partitions are chosen and the dataset distribution, some partitions might not have all classes represented. Hence the classes not represented in the partition chosen as training/validation set were removed from the test set.

The 10-fold CV and 10-fold MCV generates 10 performance measures that are averaged. The simple CV, simple MCV, nested MCV yields only one measure and thus is expected to have high variance. To mitigate this variability, training and evaluation were done five times with reinitialized networks to give a more stable measurement, thus the performance was determined by averaging these five measurements.

4.5 Network performance evaluation

The network performance was evaluated by a simple CV on the full dataset, and this performance is further referred to as the *baseline performance*, which is presented in Section 5.2. To detect dataset bias, multiple CVs were employed and the occlusion method was used to evaluate if the network has learned useful features.

The CVs were done in test scenarios, with slight variations in methodology due to differences in purpose of the tests - they could be performed to 1) detect dataset bias and/or evaluate whether the network has learned useful features; or 2) minimize the generalization error or maximize performance.

The ability to generalize was evaluated as the nominal difference between validation and test set accuracies (generalization error, GE). However, the GE in itself tells very little about bias. For example, reaching a validation accuracy of 0.2 and a test accuracy of 0.2 would give a zero GE, but if training or validation accuracy is much different from the baseline performance, this might also indicate bias. In order to ensure that the network had not been overfitted, the training accuracy were also considered. Hence in the results, the training, validation, test set accuracies are presented along with the GE. Furthermore, better performance on validation sets than similar cases might indicate bias since the network finds more patterns to base the prediction on.

The following sections describe the test cases and a brief speculation on how the performance will be affected and what would mean. Each test case has a brief explanation to motivate the test design. Due to the outcome of the test described in Section 4.5.2 (results presented in Section 5.2.2), test cases in Section 4.5.3-4.5.9 used one range bin input images if not otherwise specified.

4.5.1 Test: Full dataset cross-validation

A 10-fold CV, 10-fold MCV, and simple MCV was performed to detect bias in the dataset.

4.5.2 Test: Reducing input size

To evaluate the importance of the input dimension and whether the network has learned useful features, the number of range bins was reduced and the occlusion method was used. This can be summarized as:

- The number of range bins in the input images were reduced from 21 to five, three, and one range bin and evaluated with a simple MCV.
- Occlusion were applied to two images with size 21 range bins, one from the training set and one from the testing set in the simple MCV process.

Detailed description
Since the targets will only cover a few pixels, there is a lot of "background" in each RD-map - information in some range bins are unlikely to contain target information, such as those far away from the target. By reducing the number of range bins of the input images there should be less background for the network to learn. If the network had learned background instead of some useful feature, the training and validation performance should decrease as the number of range bins decreases since there would be fewer input dimensions to find unuseful patterns in. The GE should stay the same or decrease as the number of range bins are reduced.

The occlusion method was applied on two images with 21 range bins, occluding a patch of 11 doppler channels and three range bins. The values in the patch were set to the smallest value found in the image. The patch was applied one time on every possible location of the image. Close to the edges, only portions of the patch

was used since the whole patch was not possible to fit within the image, in order to produce a new image of the same size as the RD-map. Hence in the corners of the image, only two range bins and six doppler channels were occluded.

The images to apply occlusion on were chosen as samples that the network is very confident in (its output probability of the correct class was over 0.99), one sample from the training set and one from the test set. The probability of the correct class was measured for each step in the occlusion process. If the network has learned useful features, the class probability should not be affected very much when areas far from the target are occluded.

4.5.3 Test: Coarser class types and multiple radar systems

The importance of the class detail level was evaluated by only considering jet- or propeller aircraft and no target as classes. Adding data from another radar system were done to minimize GE. Hence the test was carried out in the following way:

- Simple MCV with coarser classes.
- Simple MCV with coarser classes and adding training data from another radar system.

Detailed description
Using the ICAO designator type as aircraft class might be unsuitable for a classification task - unique aircraft of the same class may have too different micro doppler signal and hence making it unwise to consider them the same class. Poor network performance may be because there are greater variances within the class than between classes. Hence new classes were created instead, consisting of jet aircraft, propeller aircraft, or no target, and the nested MCV were performed. If the performance or generalization increases significantly, this could indicate that the ICAO Type Designator is too detailed to be able to distinguish in the radar echo.

To minimize generalization error and maximize performance, data from a different radar system was added to the training and validation sets when performing a simple MCV. This was done when using the coarser class types since the dataset from the other radar system did not contain the ICAO Type Designator. This dataset was available at the start of this thesis thus not collected using the aforementioned data acquisition method.

4.5.4 Test: Adding temporal information using LSTM

A target is often tracked over multiple timesteps and which enable the use of RNNs. An LSTM unit of 128 units were added before the last layer, with 10 images in each time series. Is it helpful for the network to use old information from samples when classifiying the current? In [5], generalization improved greatly when adding temporal information to a micro doppler classification task, hence it might be suitable to use in this application also. The LSTM was added in the same way as in [5].

To minimize generalization error and maximize performance, the samples were aggregated in time series and evaluated in a simple MCV.

4.5.5 Test: Pruning the network

To improve generalization, pruning from 30% to 80% of the weights were done. Using two different input sizes, the networks were pruned and evaluated as:

- Simple MCV for each pruning level when using 21 range bin inputs.
- Simple MCV for each pruning level when using one range bin inputs.

Detailed description
Network pruning may have a positive effect on the network performance [22], though most pruning research uses rather large network architectures. What happens when pruning is employed in this context? Will it increase generalization? Can pruning be used to increase generalization on relatively small networks? Better generalization may indicate that pruning makes the network "unlearn" some unuseful features, but decreased generalization could indicate that the network has too few parameters to benefit from pruning.

The networks which where trained on input data with 21 and one range bin respectively in Section 4.5.2 were pruned from 30 % to 80 % during 10 epochs. This was done by the iterative method proposed in [12], which uses the magnitude of the weight as saliency measure. The performance of these networks were evaluated with the nested MCV. If the generalization error decreases as the network is pruned, this would indicate that the the network unlearns unuseful features. However, too much pruning might make the network behave strangely as too many activations becomes zero.

4.5.6 Test: In-flight radar data

To more accurately investigate real-world performance, the network was trained on data recorded in the testing facility, and tested on data that was recorded during flight containing an Airbus A320 aircraft. The whole dataset was split into a 90:10 training and validation split while the test set consisted of 150 samples recorded during flight in 2013. These samples had a rather different background since they were recorded at a different location, height and while the aircraft was moving. Good performance on the in-flight data would suggest that the learned features are robust, while bad performance indicates that the recorded data is not similar to data recorded while flying.

4.5.7 Test: Angle test

As mentioned in Section 2.2.4, radar targets emit a varying degree of micro doppler depending on their aspect angle and what kind of vessel it is. Jet aircraft creates the most micro doppler return as they fly towards the PDR, thus the network should

perform better when training and testing on jet aircraft that flies towards the PDR if it had learned micro doppler signatures.

A nested MCV was performed when the first partition only contained samples in which the aircraft fly towards the radar or within ±15 degrees aspect angle. In the second partitioning, a simple CV was used.

To make a fair comparison to the actual gains of choosing targets with small aspect angle, a simple MCV was performed with only the classes represented in the small angle set, but when keeping samples with large aspect angles.

4.5.8 Test: Restrict to data acquired by the TWS mode and downsampling

The STT radar mode is expected to produce samples that are very similar to each other since they are gathered at a faster sampling rate than of the TWS mode, yielding time correlated samples. This test aimed to determine whether the STT mode yields more biased samples than TWS, and if the TWS samples are strongly time correlated aswell. This was done by:

- Nested MCV. First partitioning removed all samples gathered in the STT mode. Then a simple MCV was performed.
- Nested MCV. First partitioning removed all STT samples and the remaining dataset were downsampled by a factor five. Then a simple MCV was performed.

Detailed description
The STT mode quickly generates large amounts of data since there is no time wasted on looking for other targets. Between each sample, the target will not have moved very far, thus the samples close in time would look rather similar.

If the generalization error decreases as the STT samples are removed, the STT samples are likely more time correlated than the TWS samples. If the generalization error further decreases when downsampling, this suggests that the TWS mode yields time correlated samples aswell. The performance was compared to the results from the one range bin case in Section 5.2.2.

4.5.9 Test: Sensitivity to additive white gaussian noise

To evaluate if clutter or noise are features that the network learned for classification, white gaussian noise of several standard deviations was added to each sample in the dataset. The test was constructed in the following way:

- Simple CV with additive gaussian noise of standard deviation 5.
- Simple CV with additive gaussian noise of standard deviation 10.
- Simple CV with additive gaussian noise of standard deviation 15.

Detailed description

There are a multitude of sources of noise and clutter in the radar echo, and if the network has learned features found in clutter or noise to base its classification on, the performance should degrade rapidly as more noise is added. This assumes that the clutter and noise levels are rather low.

Adding white noise on the input images should make the classification task harder. If the network has learned robust features in the radar signal, its performance should not be affected very much when adding a bit of noise. During this test, white gaussian noise with standard deviation 5, 10, and 15 was added to all of the samples and evaluated with a simple CV.

5
Results

In this chapter, some interesting results are presented along with a brief analysis aiming to provide possible explanations for the reasons behind the results. These explanations then provides the basis for the conclusions drawn in the following chapter. Section 5.1 presents the dataset class distribution while each Section in Section 5.2 shows the results from a test case described in Section 4.5.

5.1 Data distribution

The data was recorded on three separate occasions, the 21 February, 12 March, and 19 March. In table 5.1, some conditions during the recording days are presented.

Table 5.1: Number of RD-map samples and weather conditions during each recording day

Date	21 Feb	12 March	19 March
Weather	Rain	Rain	Sun
Samples	10 600	64 036	46 064

The number of unique ICAO Type Designators in the data was 18, and one class resembling no target was added, hence the total number of classes was 19. In Figure 5.1 the distribution of the classes can be seen. Each number corresponds to an aircraft type in Table 5.2. This shows that the data is biased in the sense that all classes are not equally represented.

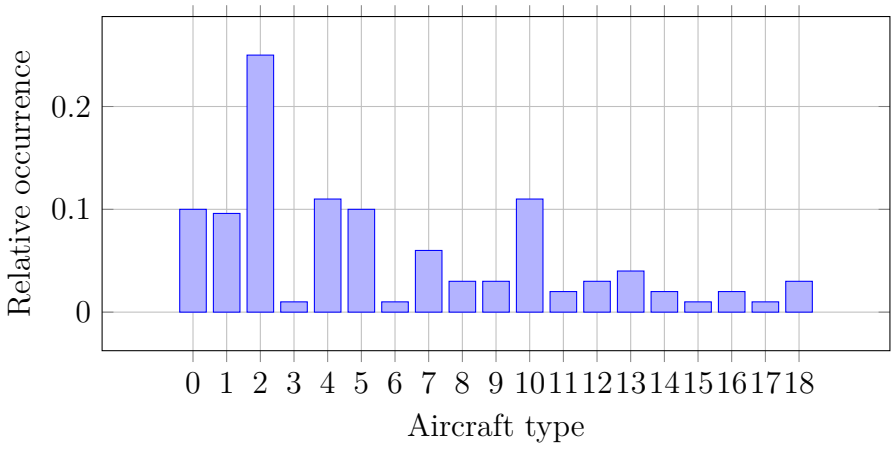

Figure 5.1: Distribution of aircraft types in the full dataset.

Table 5.2: Mapping between number and ICAO Type Designator

Aircraft type	ICAO type designator
0	No target
1	B77W
2	B738
3	A359
4	AT76
5	A320
6	B788
7	HA4T
8	A319
9	A321
10	BE20
11	A35K
12	P28A
13	E190
14	LJ75
15	A20N
16	B77L
17	B789
18	A21N

5.1.1 Class distribution on different recording occasions

As previously mentioned in Section 4.3, the simple MCV were chosen such that it split data based on the recording occasion. In the simple MCV, the data recorded in February was selected as the test set and the rest for training and validation. For this split, all classes were not represented in the training, validation and test sets. Only five classes were represented in all of the subsets, which were 1) No target; 2) A320; 3) AT76; 4) B738; 5) B77W.

Only considering these five classes, the class distribution in the subsets can be seen in Figure 5.2 and Figure 5.3. There were 65 713 and 8 942 samples recorded in the training/validation and test sets respectively when only considering the aforementioned classes.

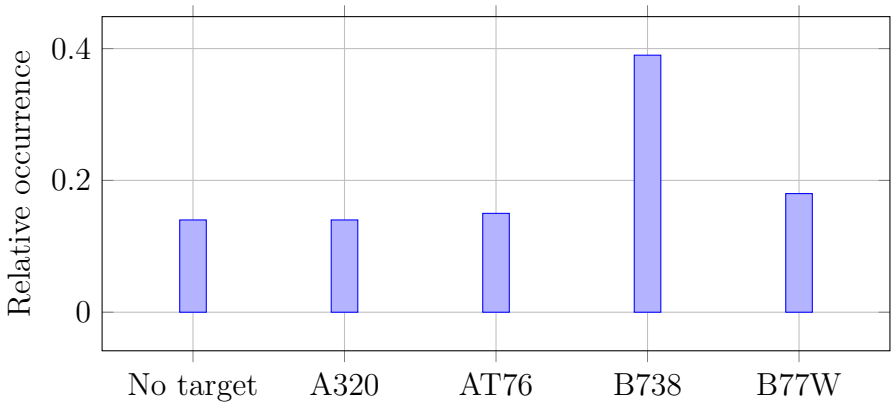

Figure 5.2: Distribution of aircraft types in the training and validation set, from both occasions in March.

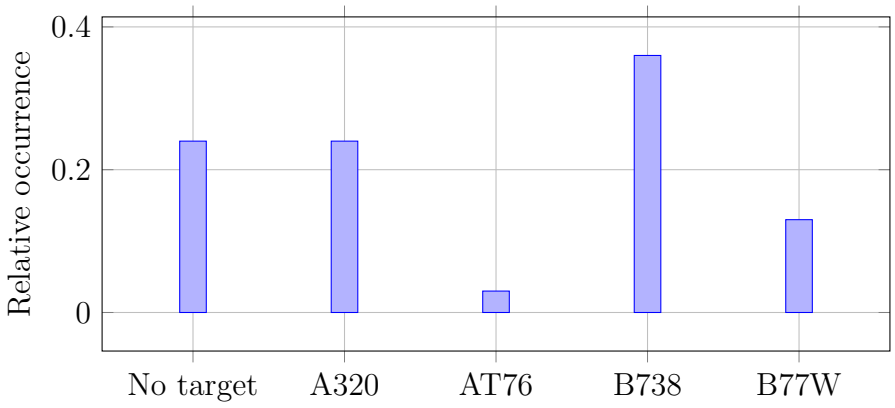

Figure 5.3: Distribution of aircraft types in the test set, recorded in February.

Thus the distributions differs somewhat; the B738 aircraft represented almost 40 % of all the samples in both sets, while the other classes differed slightly.

5.2 Test results

Table 5.3 shows the baseline performance.

Table 5.3: Baseline performance as the network accuracy on the training, validation and test sets created from a 81:9:10 split of the full dataset and 19 classes.

Case	Training	Validation	Testing	GE
Baseline	0.77	0.80	0.80	0.00

This was a rather expected result. The test set performance is similar to the validation performance which is a good sign, suggesting good generalization.

5.2.1 Results from: Full dataset cross-validation

Table 5.4: Mean performance as accuracy over the training, validation and test sets respectively.

Case	Training	Validation	Testing	GE
10-fold CV	0.77	0.80	0.80	0.00
10-fold MCV	0.78	0.81	0.27	0.54
Simple MCV	0.90	0.90	0.39	0.51

The 10-fold CV does not seem to indicate any bias since the generalization error is zero while having performance similar to other cases on training and validation sets. The 10-fold MCV and simple MCV cases however indicates that there is bias, due to high generalization error. This shows that the 10-fold CV fails to detect the bias that is detected by the 10-fold MCV and simple MCV scheme.

Since the parameters splitting the partitions in MCV are known, the 10-fold MCV indicates a strong time correlation between the samples and the simple MCV shows that there is some bias correlated to the day of the recording. This could for example be the weather during recording or that the samples are recorded close in time, but not possible to pinpoint exactly by only considering this information.

5.2.2 Results from: Reducing input size

The performance on the simple MCV when the number of range bins are reduced, are presented in Table 5.5.

Table 5.5: Performance as the mean accuracy over the training, validation and test sets respectively in a nested MCV scheme. The generalization error decreases as the number of input dimensions are decreased, likely because there are fewer dimensions to find unuseful patterns in.

Case	Training	Validation	Testing	GE
21 range bins	0.90	0.90	0.39	0.51
5 range bins	0.90	0.92	0.43	0.49
3 range bins	0.89	0.90	0.47	0.43
1 range bins	0.87	0.90	0.51	0.39

By reducing the number of range bins, the generalization error consistently decreases. The training and validation performance decreases slightly, but the test set performance increases significantly more.

In Figure 5.4, the output of the occlusion method can be seen. To the left, a sample was used which the network had been trained on, and the right is the result when using a sample from the test set.

To the left, it can be seen that the network more often reacted very little to occlusions, which is a desirable behaviour as the image contains information that is unlikely derived from the actual target (i.e. far from the center range bin). The rightmost figure shows that the predicted class probability dropped greatly as parts of the image is occluded, which suggests that it bases its prediction on nondesirable features, such as around range bin 20.

From both the performance measurements and the occlusion analysis, it can be concluded that the baseline performance is highly misleading - the network seems to have learned features far from the target and fails to generalize. Reducing the number of input dimensions seems to reduce the likelihood of learning unuseful features. Yet even with one range bin it is likely that the network has learned unuseful features due to the poor generalization.

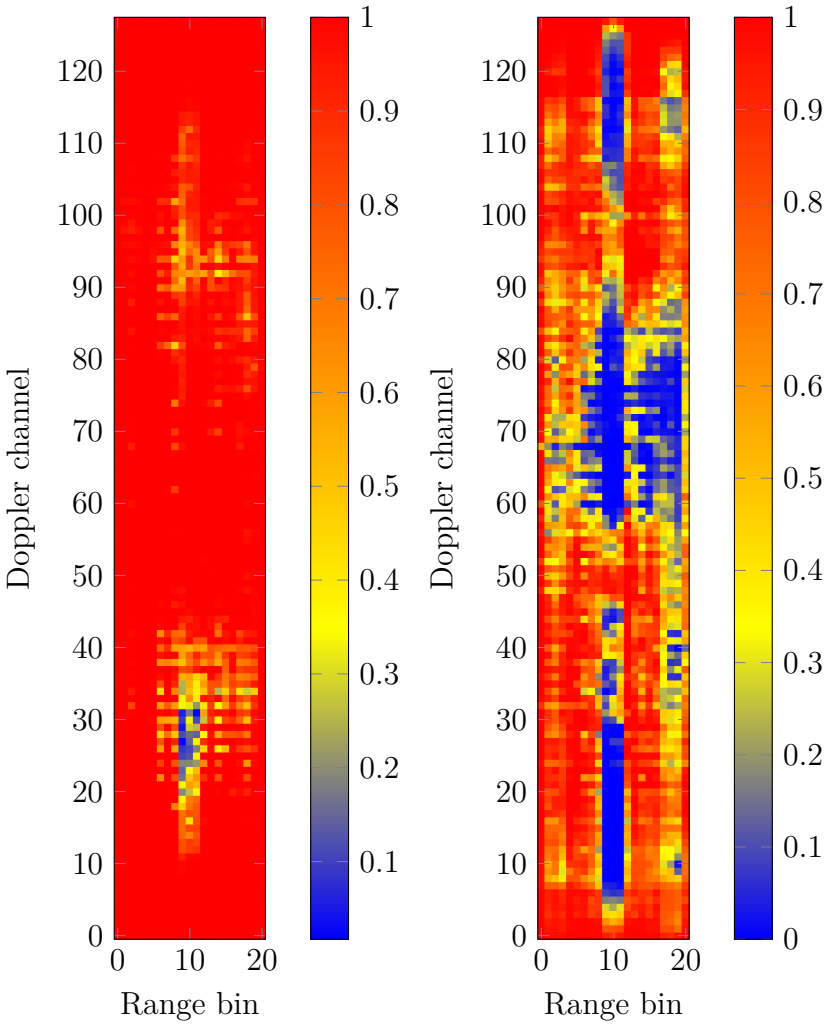

Figure 5.4: Class probability as parts of image is occluded. The target is located somewhere along range bin 10. Each pixel show how likely a given class is according to the network as a patch centered in that location of the RD-map is occluded. A probability of 0.5 means that the network assigns a 50 % probability of that sample representing a given class, and 50% probability for all other classes. To the left, an image that has been trained on acts as the input sample while to the right, the input image was a sample that the network had not trained on. To the right, the network reacted aggressively to occlusion of areas that is expected to contain very little target information, while to the left it mostly reacts very little.

5.2.3 Results from: Coarser class types, with and without ground radar data

By only considering three classes - jet, propeller or no target resulted in the data distribution that can be seen in Figure 5.5 and Figure 5.6.

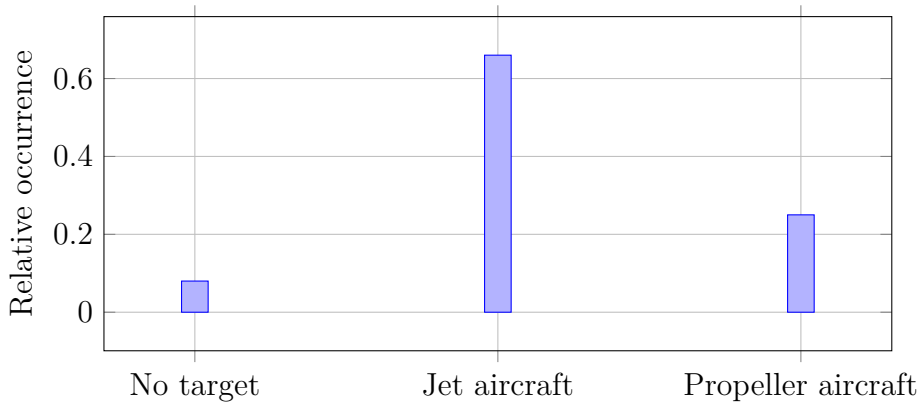

Figure 5.5: Distribution of aircraft types in the training and validation set.

The distribution of the test set is shown in Figure 5.6.

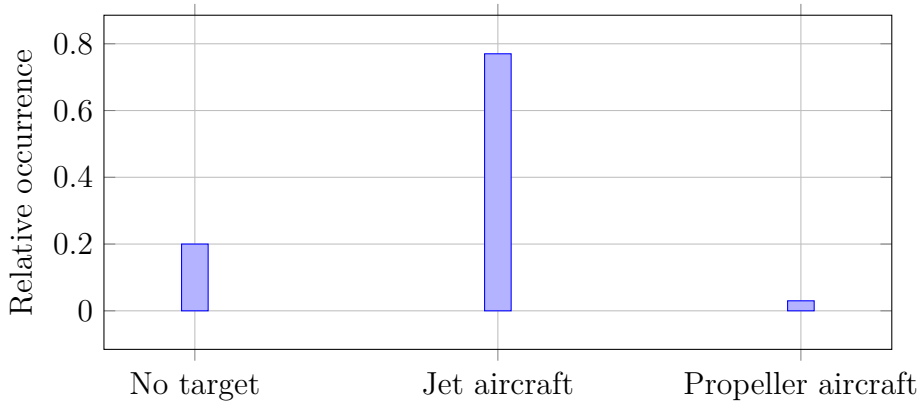

Figure 5.6: Distribution of aircraft types in the test set.

The data from the ground radar system consisted of 68 958 samples, 67 % jet aircraft and 33 % propeller aircraft, not containing the no target class. The results from the simple MCV can be seen in Table 5.6 with coarser classes and when ground radar data is added.

Table 5.6: Performance on simplified problem formulation as accuracy over the training, validation and test sets respectively.

Case	Training	Validation	Testing accuracy	GE
No ground data	0.87	0.89	0.85	0.04
With ground data	0.82	0.83	0.87	-0.04

Generalization increased compared to baseline when choosing different classes, which was expected since this was a simpler problem. Adding data from another radar system seemed to further minimize the generalization error slightly, although this difference might not be significant. Yet a rather interesting discovery is that it did not worsen the performance, suggesting that some target characteristics may be similar even in different radar systems. Due to the low number of propeller aircraft in the test set (3%), it is inconclusive if the micro doppler signal has been learned.

5.2.4 Results from: Adding temporal information using LSTM

In Table 5.7, the accuracies from the simple MCV are presented when adding an LSTM layer . To make a direct comparison, the performance when no LSTM layer is used is also shown, as earlier presented in Section 5.2.2.

Table 5.7: Generalization accuraccies with and without LSTM

Case	Training	Validation	Testing	GE
1 range bin	0.87	0.90	0.51	0.39
1 range bin + LSTM	0.94	0.92	0.53	0.39

Thus adding an LSTM layer did not minimize the generalization error, but increased the accuracies. This can be interpreted as the LSTM layer is making the network learn better but still suffers from the dataset bias. In contrast to [5], there was no great improvement, which is likely due to the nature of the problem - an aircraft does not change type over time.

5.2.5 Results from: Pruning the network

Figure 5.7 shows the test set performance in the simple MCV as weights were pruned. Both networks seemed to increase test set performance as weights are pruned, but only to some extent - the 21 range bin network seemed to maximize its performance with 70 % weights pruned, while the one range bin network had its max when 60 % weights were pruned. This suggests that pruning can be used to improve generalization. The difference in optimal pruning percentage can be due to the difference in number of parameters between the networks.

The one range bin network had a test set accuracy of almost 40 % when pruned to 80 % - an interesting observation is that the test set contains almost 40 % of the B738 class. This much pruning is thus likely to have made the network always guess B738 for all samples in the test set.

Validation set accuracy is also of interest, since it represents how well the networks performs on samples that it has seen. In Figure 5.8, the validation set accuracies are shown for the same pruning levels as previously.

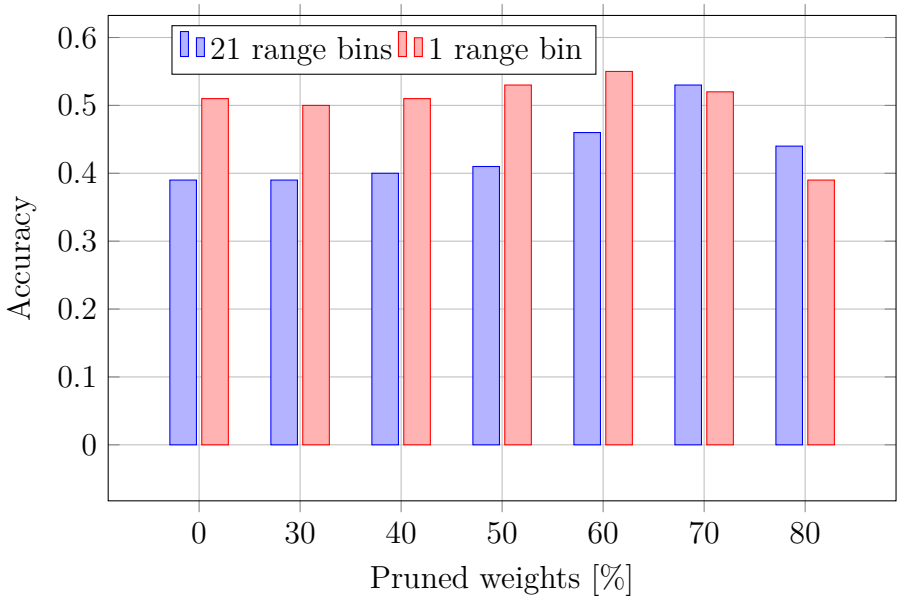

Figure 5.7: Network accuracy on test set for different pruning levels. The 21 range bin network seems to benefit the most from pruning while the one range bin network was not affected as much, until a rather large pruning level had been reached.

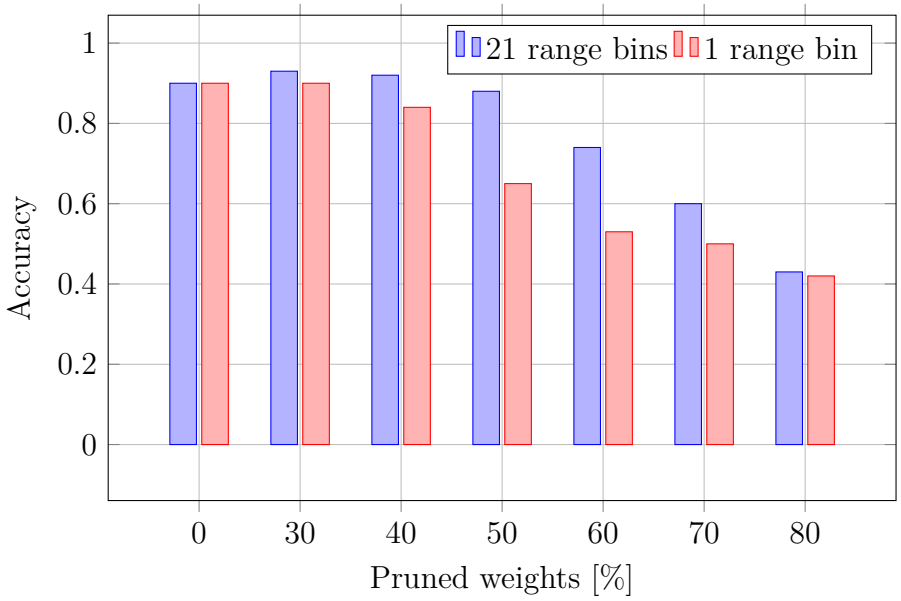

Figure 5.8: Network accuracy on validation set for different pruning levels.

The validation accuracy increases slightly at low pruning levels, but degrades as the pruning level increases. This can be interpreted as the network unlearning its trained features with pruning, since it is unable to retain its validation set accuracy. But because of dataset bias, this is expected to be good to some extent since some learned features are not useful for generalizing.

5.2.6 Results from: In-flight radar data

The network was trained and validated on the whole dataset and 150 samples from an A320 was used for testing, and produced the results seen in Table 5.8.

Table 5.8: Network performance when trained and validated on whole dataset, while testing on a few samples when the aircraft was moving.

Case	Training	Validation	Testing
1 range bin + in-flight	0.69	0.75	0.008

The network performed extremely poorly on the test set which indicates that the features learned are not very robust. The lack of training samples where the radar is moving is probably the most contributing factor but it might aswell be other factors such as the weather.

5.2.7 Results from: Small Angle

The number of samples where the aircraft had an aspect angle of less than 15 degrees were 93 361, distributed as can be seen in Figure 5.9.

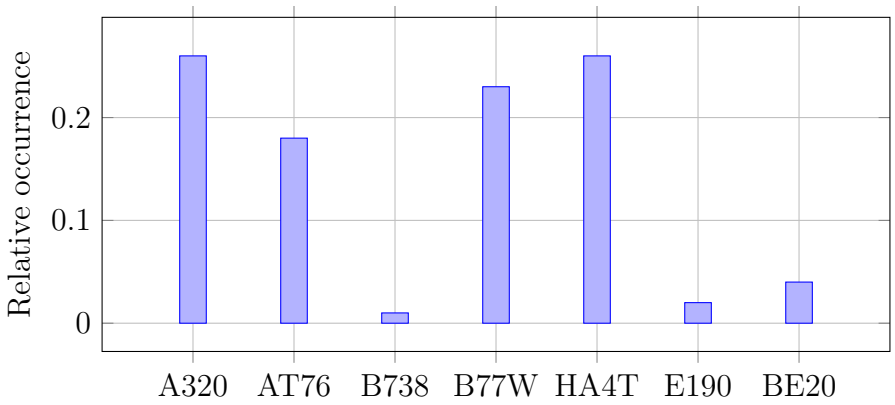

Figure 5.9: Distribution of aircraft types with an angle under 15 degrees.

The performance can be seen in Table 5.9 compared to the simple CV when only the classes represented in the small angle dataset are considered.

Table 5.9: Network accuracy when only considering targets with small angle

Case	Training	Validation	Testing	GE
1 range bin + fewer classes	0.76	0.81	0.81	0.00
1 range bin + small angles	0.85	0.87	0.88	-0.01

Considering targets with small angles appeard to be beneficial for the classification - all of the accuracies were higher when only small angle targets were used for training and testing. This suggests that the micro doppler signal is somewhat important for the classification, but to an unknown extent because it has been previously shown that the dataset is biased.

5.2.8 Results from: Restrict to data acquired by TWS and downsampling

When removing all the samples collected with the STT mode, there were 42 718 samples left, distributed as in Figure 5.10, and the mapping from numbers to aircraft type can be seen in Table 5.2.

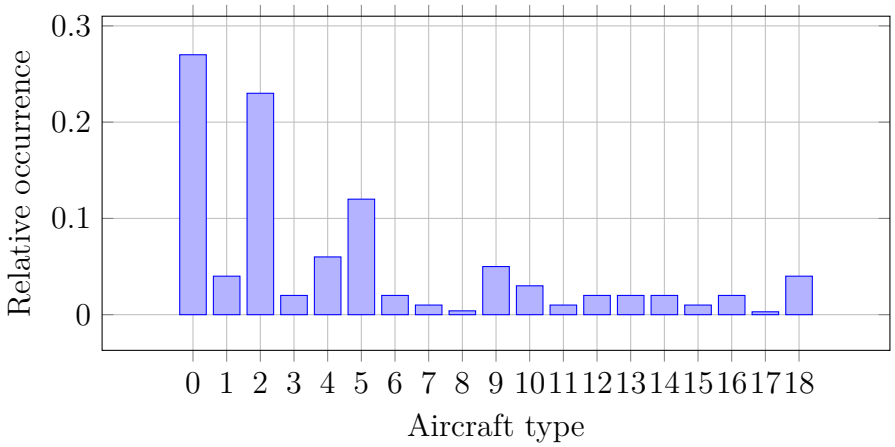

Figure 5.10: Distribution of aircraft types when removing the samples acquired by the STT mode.

Without STT samples, the performance is presented in Table 5.10 with and without downsampling, along with the one range bin performance earlier presented.

Table 5.10: Generalization performance as accuracy on training, validation and test sets when including STT data and when using only TWS. For comparison, the case ”1 range bin” is that of Section 5.2.2

Case	Training	Validation	Testing	GE
1 range bin	0.87	0.90	0.51	0.39
1 range bin + only TWS	0.63	0.65	0.37	0.28
1 range bin + only TWS + downsampling	0.58	0.53	0.37	0.16

Removing the STT samples made the network perform worse on all sets, yet decreased the generalization error. Downsampling worsened performance on training and validation sets but further decreased the generalization error, which indicates that many of the samples contribute little to learning useful features since the same test set accuracy could be reached with only 20% of the samples.

This suggests that the STT samples are strongly time correlated, and the TWS samples aswell. Thus further measures must be carried out to combat the dataset time correlation in order to create a less biased dataset, such as downsampling.

5.2.9 Results from: Sensitivity to additive white gaussian noise

Adding white gaussian noise to the input samples seemed to have quite an effect on the test set performance, as seen in Figure 5.11. Accuracies on other sets were not presented here since the test set performance were always very similar to the training and validation performance when using simple CV.

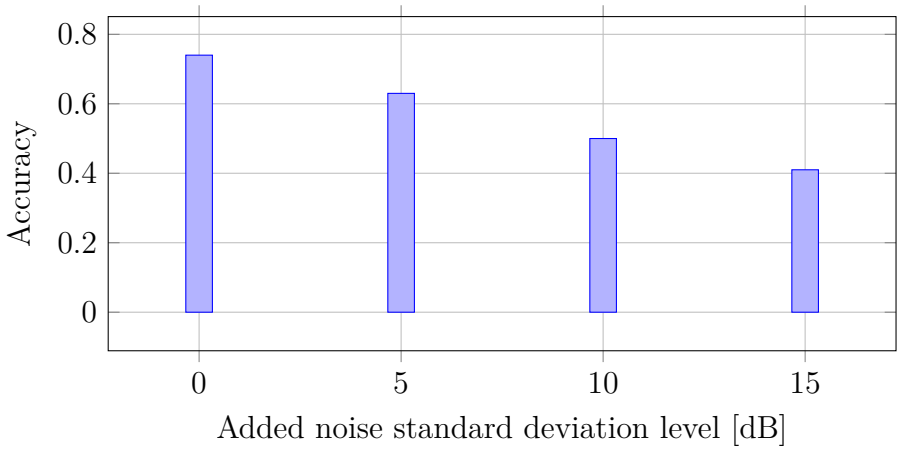

Figure 5.11: Test set accuracy when adding white gaussian noise

The performance seems to degrade rather linearly with increasing added noise. This sensitivity can be interpreted such that the network still has learned unuseful features, since the performance degrades quite rapidly. It is also possible that the network has found useful features, such as micro doppler signature, but that they are rather weak and is too distorted by the noise to be detected by the network. None of these explanations seem more plausible since it is not known if these noise levels are reasonable or not.

5.3 Summary of test results

Performing a 10-fold CV did not show any sign of dataset bias, but the 10-fold MCV indicated a sample time correlation. The MCV indicated that data recorded on the same occasions were rather similar, since generalizing across occasions were troublesome, further supporting the hypothesis of time correlated samples. The time correlation is believed to be derived from the fact that samples were recorded close in time, such that the target had not moved very far between samples. The weather conditions were also expected to play a significant role.

The time correlation between samples was also detected when performing the occlusion analysis as the network seemed to have learned unuseful features in the background to base its prediction on. Reducing the samples dimensions minimized the generalization error because there were fewer dimensions to find unuseful patterns in.

The aircraft classes were rather detailed as represented by its ICAO Type Designator, and when choosing less detailed classes as 1) no target; 2) jet aircraft; and 3) propeller aircraft the generalization greatly improved, suggesting that the ICAO Type Designator may be too detailed to distinguish between targets in a radar echo. Yet this improvement was inconclusive since the coarser test set was very imbalanced, consisting of almost 80 % jet aircraft. Adding data of jet- or propeller aircraft from a different radar system did not significantly increase generalization performance, but there was no degradation either, suggesting that data from different radar systems is possible to combine.

Using several samples in a time series did not significantly improve generalization performance. In [5], the classification got better when adding a LSTM layer, but the success was likely due to the nature of the problem - the state of a cooking process may change over time but in this case, an aircraft type does not change over time.

Network pruning seemed to increase the generalizability of the network, thus may be used for reducing overfitting or unlearning patterns.

The network was evaluated on samples gathered from a real-world scenario as the radar system was in flight, and performed extremely poor, suggesting that the data collected in-flight is too different from that collected from the test facility. This was believed to be mainly because of the differences in how clutter appears in the RD-maps, but also because of the training data bias.

Using only targets that were recorded while flying towards the radar increased performance on all sets. This suggests that the micro doppler signature is used for classification, but to an unknown extent due to dataset bias.

Removing STT samples resulted in a lower performance on all sets but decreased the generalization error, suggesting that the STT samples were strongly time correlated. When also downsampling the remaining partition, the performance decreased on the training and validation sets but not on the test set, thus the generalization error was further reduced. From this it was concluded that removing STT samples did not remove all time correlation bias - the TWS mode also generates time correlated samples. Therefore the STT mode should be especially avoided during recording, but downsampling or other measures should also be carried out to mitigate the sample time correlation.

Adding white gaussian noise to the input data impaired the performance significantly, but this could both indicate that the network has learned useful and unuseful features. Thus that test case did not yield any useful information.

6
Discussion

The following discussion in this chapter uses sources of errors and the overall implication of the results to identify criterias that could be used for assessing the feasibility of the machine learning filter on the radar data.

6.1 Strength and weaknesses of methods

All the tests were carried out with the same network architecture. When changing the input sizes or classes, it is likely that a different network architecture would yield a different result. But with this in mind, performance differences needed to be large to be considered significant. Therefore, it is not believed that optimizing the network hyperparameters would have had an impact on the conclusions of this thesis.

Analysis of the results were based on the accuracy of the networks but not on the loss, which might have been used for a more in-depth analysis.

Network pruning had a positive effect on the generalization, although the method used does not have any effect on the memory usage or computational complexity of the network since it was an unstructured method and no compression was used. If the computational load needs to be lowered, then it is possible to resort to other pruning methods.

During the timeline of this Master's thesis, the outbreak of the coronavirus disease (Covid-19) had a great impact on the amount of commercial aircraft flying over the Gothenburg region [47]. Collection of data became cumbersome since it was not efficient to keep the testing facility up and running whilst awating the very few aircraft to pass by. Thus data was recorded fewer times than predicted and the trial-and-error approach was not as powerful. The data collected was analyzed but it was not possible to collect new data upon new observations as efficiently needed for the timeline of this thesis or to counteract the discovered and potential sources of bias. Having more data or recordings from more separate occasions would have made the results more conclusive.

In the process of automatic labelling, 21 range bins were selected around the target. However, the radar system did at times provide several target detections, of which then the detection with strongest amplitude was chosen. This may have led to that the wrong detection was selected for some samples, which could have been noise or

a completely different aircraft. It could be necessary to use a more sophisticated method for associating the targets in each RD-map with ATC information.

It is worth noticing that all results are performed on data collected using TWS and STT mode, which often use medium PRF during operation. Using a higher PRF could provide more details in the RD-maps and thus potentially improve classification performance.

6.2 Useful features

Analysis of the test results indicated that the dataset was biased, but also that the network uses micro doppler signature as basis of classification. Due to the dataset bias it is not possible to say how important the micro doppler signal is for the classification. When the number of input dimensions were reduced, the strong time correlation between samples seemed to persist even as there were fewer dimensions to find nonuseful patterns in. From this it is believed that the network mostly had learned nonuseful features, and that the micro doppler signal was used to a lesser extent.

Since all training data was gathered from a stationary radar, there was one source of bias not accounted for. As stated in Section 3.3, the training data must be similar to the data generated in the target environment, but in this case the target environment is a flying aircraft. Thus one source of bias is the fact that all the samples were stationary collected, which will have a significant impact, as was seen when testing on data recorded in-flight in Section 5.2.6. From this it is concluded that the network had not learned the micro doppler signatures sufficiently well to perform good on in-flight data.

Hence this data acquisition method cannot be used for creating algorithms that will be run in a flying radar. If the bias from time correlation can be mitigated, the next problem will be mitigating the bias inherited from collecting data while stationary. This is expected to be possible by data augmentation, where the clutter in a flying radar can be calculated and added to the samples.

The weather conditions are expected to be a source of bias, but it was not considered as a splitting parameter in an MCV. It is possible that the weather condition is the most significant source of bias, making the samples time correlated.

As shown in Section 5.2.8, when downsampling the dataset, the same test set accuracy was reached while also minimizing the generalization error. This can be interpreted as the number of training samples does not need to be that many, but that it is more important that the samples contain more variation.

6.3 Feasibility of machine learning in airborne radar

Filters in radar systems are "traditional learning" which is based upon theory that explains how the world works. Machine learning is theoryless and learns from experience by finding patterns in data, without any perception of how the world works. Therefore, machine learning cannot work without great amounts of data with proper information. Thus determining the possiblity of aggregating large amount of data in a meaningful way provides answer whether or not machine learning is feasible in the airborne radar.

To begin with, it was simple to collect and label great amounts of data using an airborne system. However the techniques that made it simple also induced misleading results. The dataset became too homogenous since the samples were to much alike, causing bias. Sampling with STT proved to be a bad idea as it did not improve classification performance even if it enabled gathering greater amounts of data. Using an LSTM layer and time series of data did not improve generalization as was believed, but likely because the samples were so similar that adding more did not add any information. A vital tool used to detect dataset bias was the MCV scheme.

The main source of bias is likely the clutter in the RD-maps. When representing radar data as RD-maps the clutter creates patterns in the images that are interpreted as features and thus becoming significant to the network. This is based upon the minimized generalization error from removing almost all range-bins and the occlusion analysis, but also the terrible performance when using a network trained on stationary data with in-flight data.

The tests also show how to counteract the impact of clutter in the images. Network pruning show potential to counteract bias in the dataset, by unlearning less signficant features. Furthermore, a different approach to classes increased the accuracy to 88 %, without any work on the network itself. Thus indicating the potential of using RD-map classification for detection: target or no target. The possibility of combining data from different radar systems is very interesting. Using a data-driven approach it appears to be easy to combine data from different systems. Since it does not require any modelling of the systems, only data with similar formats.

In the timeline of this thesis it was thus possible to collect and label almost 200 000 datapoints where approximately 125 000 were used in the final dataset. Data with basic signal processing methods (absolute value, FFT and an arbitrary window function) in combination with a simple neural network reached an accuracy of 88 % percent on course class types, and 87 % in separating five aircraft types in a subset where the aspect angle was small. Which indicates the feasibility of collecting a large and useful dataset . Both tests mentioned above was validated with the method for detecting misleading results. However, it was not as trivial as simply collecting a large amount of data. A clear strategy is necessary to avoid misleading results, the MCV splitting the data between month of recording. Thus the airborne radar system's capability of creating a lot of data endorses the feasbility of machine

learning, and the possibility of classifying aircraft emphasize that there are meningful patterns to learn from.

Deep learning algorithms are also expected to be feasible from a computational load perspective. Traditional radar signal processing algorithms are computationally intensive, hence neural networks are expected to be possible to run in hardware designed for radar signal processing. As shown in this thesis, there exist pruning methods for lowering the computational complexity of neural networks which can be used if the developed algorithm cannot fulfill the computational constraints.

Arguing in favor of a theoryless method using the following principle might be a humorous contradiction, nevertheless becoming that "the most simplest solution is most likely the right one" - the Occam's principle states. Thus "the simplest model that fits the data is also the most plausible" [48]. Lending the reasoning from [48] where a simple candidate, in this case a lightweight neural network, interpreting the intended datset is more meaningful than a complex solution. Considering all results, it is generally not a big difference between the training and validation sets. Furthermore, the training set often produce good performance. Indicating that there is no need for a bigger network with more weights/layers. The simpler network is sufficient, but the composition, labelling and content of the dataset must still be investigated. An RD-map does not contain as much detail and thus not as much information as an optical image. This means that a small CNN, with regards to trainable parameters, is likely sufficent.

Furthermore, in order to use supervised learning and produce RD-maps the full data- and signal processing chain of the radar system is necessary, since the data was automatically labelled by fusing tracking data from the radar with ATC data. Metadata is valuable for verifying network performance and finding dataset bias. This work did not perform any attempts with raw IQ data or without metadata. Thus this work could not replace the data- & signal processing chain but work together with it to possibly add new or helpful functionality.

In conclusion, there are three criterions used in this discussion to determine whether or not machine learning is feasible in airborne radar systems. The challenge of collecting a great amount of data is considered overcome by this work. Network pruning and the fact that it is not necessary to use a large network to learn from RD-maps provides an example of how machine learning could work under constraints of a flying system. However, one criterion remain inconclusive, namely verification. This work have succeded to verify that the results can be misleading when using the collected dataset, by suggesting a variety of tests to perform on a trained network. Thus the challenge of verifying the results are commenced, but it is not possible to determine a secure method of verification since no sufficently unbiased datasets were discovered. Thus this work addresses the challenge of verifying the results rather succesfully, but since no results could be trusted, this work verifies that the network would not have good performance in a real-world implementation. Should then machine learning be considerd infeasible for an airborne radar system? This

work suggest that datacollection is not trivial when using an airborne radar system, and common machine learning techinques cannot be applied without adjustments. However, since it was possible to succesfully create methods for each criterion during the timeline of this work, machine learning could be considered feasible if techniques to counteract the biased dataset is researched.

7
Conclusion

In this thesis, the feasibility of data-driven algorithm development in radar signal processing has been investigated. Many aspects of implementing deep learning are considered feasible, such as accessing large quantities of data and computational complexity. However, real-world implementation could not be deemed feasible by this work, since the data collection method generated biased samples. Due to this bias, it is not possible to verify that the algorithm will exhibit desirable behaviour in a real-world implementation, thus data-driven development will not be feasible until this is solved.

Radar data was represented as images and a CNN was trained to determine radar target classes, with good performance. A 10-fold CV scheme was employed to evaluate data quality but did not suggest that the dataset was biased. However, the proposed 10-fold MCV scheme suggested that the samples had a strong time correlation, which means that the initial network performance was not representative of a real-world scenario.

Occlusion analysis suggested that the network had learned unuseful features, which further indicated that the dataset was biased. Reducing the number of input dimensions helped to decrease the generalization error since there were fewer dimensions to find unuseful patterns in, yet this did not completely remove the effects from the dataset bias. Network pruning also helped in reducing the generalization error but only to some extent as performance dropped rapidly when too many weights were removed. It was concluded that the network had not learned micro doppler signal sufficiently well to be able to perform well on real-world data, due to dataset bias. The computational load of neural networks are not expected to be problematic since traditional radar algorithms are computationally intensive as well.

The proposed data collection method can be used to quickly generate large amounts of biased data, which must be accounted for. Deep learning algorithms are believed to be potentially useful in radar signal processing, but a core issue is to find and mitigate the bias introduced by the collection method. Humans must be able to trust the network before they are willing to implement it in a real system, and using methods from Explainable AI will be an essential way of determining whether the algorithm will behave as expected or not. The proposed MCV schemes are expected to be useful for detecting bias in order to adapt the data collection process, even for collecting data for other functions using an airborne radar. The greatest obstacle in applying deep learning in radar applications is to develop a method for creating an

unbiased dataset and to verify that the algorithm will behave as intended.

7.1 Future research

Due to the challenges and discoveries of this thesis there exist several aspects that would need further investigation. RD-maps are less detailed than optical images, thus it is reasonable that there is less information to extract hence reducing the need of complex network architectures and number of training samples. This limits the amount of target information that could be extracted from an RD-map but is expected to make the network train faster and/or on fewer samples than in applications using optical images. It would therefore be interesting to evaluate what sample size it would require to develop a deep learning algorithm on radar data in comparison to other fields such as image analysis. Because data can be gathered cheaply at a low quality, would it be better to more carefully choose the dataset than to just gather more low quality data?

Data augmentation is commonly used in image analysis but was not employed in this thesis. What kinds of data augmentation could be employed on radar data represented as images and what would its effects be on network performance and robustness? This could for instance be adding a clutter profile to simulate a moving radar.

Since the sample size from a moving radar system was very small, a future study could evaluate the performance on data from a moving radar when the network is trained on stationary data. Then how the performance is affected by using data augmentation could be evaluated, to find a suitable augmentation method.

In this thesis, the radar data was represented as images, but other representations might be more suitable for machine learning problems. Thus the different representations of radar data could be evaluated to find the most promising representation.

Target detection is a key functionality of many radar systems, and shows potential to be carried out by deep learning algorithms. Wang *et al.* [6] used purely synthetic radar data and Brodeski et al. [37] used calibration data, hence it would be interesting to use the data collection method from this thesis to investigate target detection on real-world data. The algorithm presented in [37] is based on the Faster R-CNN network architecture, but the Faster R-CNN is considered slow in comparison to the YOLO algorithm [36]. Implementing radar target detection with YOLO instead could be the aim of future research.

Radar target classification using supervised deep learning algorithms requires a dataset containing the signatures of the targets that shall be detected. In military applications, it is often impossible to acquire data of the targets of interest, such as enemy aircraft, making supervised approaches obsolete. Thus investigating the potential of unsupervised learning for radar target classification would be suit-

able for future research. This could for instance be training a network to distinguish between targets that have already been seen (such as friendly or commercial aircraft) and those that have never been seen before (potentially enemy aircraft) using unsupervised anomaly detection.

The MCV scheme could be further elaborated, it proved very useful to investigate bias. For instance, if there exists no metadata of the dataset, would it be able to create "synthetic" metadata from information contained in the data itself? For large dimensional samples it could be calculating mean or variance in each sample, then splitting data based on these parameters.

7.1.1 Ethical aspects

Radar systems have traditionally been an expensive technology mostly used in military application, but as the technology becomes cheaper it becomes a more viable choice in other fields. The work presented in this thesis is focused on military usage but can also be applied to civil radar development, although this data collection method is typically only viable for long range radar systems.

Deep learning may prove a cheaper option to traditional radar algorithm development, making it possible for smaller organizations to integrate radar technology into their products. Then radar technology might be feasible to use in other types of applications without requiring thorough radar knowledge.

The usefulness of deep learning algorithms are greatly decreased because of their black box nature, and the proposed MCV scheme might be useful for finding dataset bias in other fields. This would be useful to detect potentially dangerous, discriminative or other undesirable behaviour of deep learning algorithms before they are implemented.

In military applications, actions based on incorrect information can have devastating consequences, making it very important that algorithms implemented in these applications are robust and can be trusted sufficiently. Thus it is more important in this sector to be able to verify that the algorithms will not behave in an undesirable way. Radar systems are also an integral part of the defence capabilities of any modern nation but are traditionally expensive. Cheaper radar systems may reduce public military spending, and enable poorer countries to defend themselves or at a lower cost. However, more accessible radar technology may enable using radar for destructive purposes.

Bibliography

[1] G. W. Stimson, *Introduction to Airborne Radar*. Mendham: SciTech Publishing inc, 1998.

[2] Directorate-general for Communication networks, content and technology (CONNECT), "Shaping Europes digital future: Member states sign up to cooperate on Artifical Intelligence," 2018, accessed: 2020-05-13. [Online]. Available: https://ec.europa.eu/digital-single-market/en/news/eu-member-states-sign-cooperate-artificial-intelligence

[3] E. Wallin, "Detecting Jamming and Interference in Airborne Radar Using Convolutional Neural Networks," Master's thesis, Chalmers University of Technology, 2019.

[4] H. Andersson and C. T. Luong, "Classification between birds and UAVs using recurrent neural networks," Master's thesis, Chalmers University of Technology, 2019.

[5] M. Altmann, P. Ott, N. C. Stache, and C. Waldschmidt, "Learning Dynamic Processes from a Range-Doppler Map Time Series with LSTM Networks," in *2019 16th European Radar Conference (EuRAD)*. IEEE, 2019.

[6] L. Wang, J. Tang, and Q. Liao, "A Study on Radar Target Detection Based on Deep Neural Networks," *IEEE Sensors Letters*, vol. 3, no. 3, pp. 1–4, 2019.

[7] D. Montgomery, G. Holmén, P. Almers, and A. Jakobsson, "Surface classification with millimeter-wave radar using temporal features and machine learning," in *2019 16th European Radar Conference (EuRAD)*, 2019, pp. 1–4.

[8] P. Molchanov, K. Egiazarian, J. Astola, A. Totsky, S. Leshchenko, and M. P. Jarabo-Amores, "Classification of aircraft using micro-Doppler bicoherence-based features," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 50, no. 2, pp. 1455–1467, 2014.

[9] I. Goodfellow, Y. Bengio, and C. Aaron, *Deep Learning*. MIT Press, 2016, accessed: 2020-03-12. [Online]. Available: http://www.deeplearningbook.org

[10] V. C. Chen, F. Li, S. S. Ho, and H. Wechsler, "Micro-doppler effect in radar: Phenomenon, model, and simulation study," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 42, no. 1, pp. 2–21, 2006.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Conference on Neural Information Processing Systems*, 2012.

[12] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," Tech. Rep., 10 2017. [Online]. Available: http://arxiv.org/abs/1710.01878

[13] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A Survey of Model Compression and Acceleration for Deep Neural Networks," *IEEE Signal Processing Magazine*, 10 2017. [Online]. Available: http://arxiv.org/abs/1710.09282

[14] J.-H. Luo, J. Wu, and W. Lin, "ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression," in *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 7 2017. [Online]. Available: http://arxiv.org/abs/1707.06342

[15] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both Weights and Connections for Efficient Neural Networks," in *2015 NIPS*, 6 2015. [Online]. Available: http://arxiv.org/abs/1506.02626

[16] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the Value of Network Pruning," in *2019 ICLR*, 10 2018. [Online]. Available: http://arxiv.org/abs/1810.05270

[17] J. Denker, S. A. Solla, Y. Le Cun, and J. S. Denker, "Optimal Brain Damage," *Advances in neural information processing*, 1989. [Online]. Available: https://www.researchgate.net/publication/221618539

[18] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," in *1993 IEEE International Conference on Neural Networks*. Publ by IEEE, 1993, pp. 293–299.

[19] S. Wiedemann, H. Kirchoffer, S. Matlage, P. Haase, A. Marban, T. Marinc, D. Neumann, T. Nguyen, A. Osman, D. Marpe, H. Schwarz, T. Wiegand, and W. Samek, "DeepCABAC: A Universal Compression Algorithm for Deep Neural Networks," Tech. Rep., 7 2019. [Online]. Available: http://arxiv.org/abs/1907.11900

[20] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," in *2016 ISCA*, 2 2016. [Online]. Available: http://arxiv.org/abs/1602.01528

[21] K. Paupamah, S. James, and R. Klein, "Quantisation and Pruning for Neural Network Compression and Regularisation," 1 2020. [Online]. Available: http://arxiv.org/abs/2001.04850

[22] S.-K. Yeom, P. Seegerer, S. Lapuschkin, S. Wiedemann, K.-R. Müller, and W. Samek, "Pruning by Explaining: A Novel Criterion for Deep Neural Network Pruning," 12 2019. [Online]. Available: http://arxiv.org/abs/1912.08881

[23] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. K. Nvidia, "Importance Estimation for Neural Network Pruning," Tech. Rep., 2019. [Online]. Available: https://github.com/NVlabs/Taylor_pruning.

[24] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning Convolutional Neural Networks for Resource Efficient Inference," in *2017 ICLR*, 11 2016. [Online]. Available: http://arxiv.org/abs/1611.06440

[25] J. Frankle and M. Carbin, "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks," in *2019 ICLR*, 3 2018. [Online]. Available: http://arxiv.org/abs/1803.03635

[26] H. Zhou, J. Lan, R. Liu, and J. Yosinski, "Deconstructing Lottery Tickets: Zeros, Signs, and the Supermask," in *2019 NIPS*, 5 2019. [Online]. Available: http://arxiv.org/abs/1905.01067

[27] A. Torralba and A. A. Efros, "Unbiased Look at Dataset Bias," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2011, pp. 1521–1528.

[28] M. Stone, "Cross-Validatory Choice and Assessment of Statistical Predictions," vol. 36, no. 2, pp. 111–147, 1974.

[29] D. Gunning, "Explainable Artificial Intelligence (XAI)," 2017, accessed: 2020-05-20. [Online]. Available: https://www.darpa.mil/program/explainable-artificial-intelligence

[30] A. B. Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. García, S. Gil-López, D. Molina, R. Benjamins, R. Chatila, and F. Herrera, "Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI," *Information Fusion*, pp. 82–115, 06 2020. [Online]. Available: http://arxiv.org/abs/1910.10045

[31] S. Bach, A. Binder, G. Montavon, F. Klauschen, K. R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PLoS ONE*, vol. 10, no. 7, 7 2015.

[32] M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks," 11 2013. [Online]. Available: http://arxiv.org/abs/1311.2901

[33] J. Long, E. Shelhamer, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," in *2015 CVPR*, 2015.

[34] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *2014 CVPR*, 11 2013. [Online]. Available: http://arxiv.org/abs/1311.2524

[35] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," Tech. Rep., 6 2015. [Online]. Available: http://arxiv.org/abs/1506.01497

[36] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *2016 CVPR*. IEEE, 6 2015. [Online]. Available: http://arxiv.org/abs/1506.02640

[37] D. Brodeski, I. Bilik, and R. Giryes, "Deep Radar Detector," in *2019 RadarConf*, 2019. [Online]. Available: https://arxiv.org/abs/1906.12187

[38] E. T. Kouba, "Recurrent neural networks for radar target identification," Master's thesis, Austin, TX, 1993.

[39] V. Jithesh, M. J. Sagayaraj, and K. G. Srinivasa, "LSTM recurrent neural networks for high resolution range profile based radar target classification," in *2017 CICT*. IEEE, 2017.

[40] F. Gao, T. Huang, J. Wang, J. Sun, A. Hussain, and H. Zhou, "A Novel Multi-Input Bidirectional LSTM and HMM Based Approach for Target Recognition from Multi-Domain Radar Range Profiles," *Electronics*, vol. 8, no. 5, p. 535, 5 2019.

[41] S.-H. Park, "Automatic Target Recognition Using Jet Engine Modulation and Time-Frequency Transform," vol. 39, pp. 151–159, 2014.

[42] T. Thayaparan, S. Abrol, and E. Riseborough, "Micro-Doppler Radar Signatures for Intelligent Target Recognition," Tech. Rep. September, 2004.

[43] D. Yessad, A. Amrouche, M. Debyeche, and M. Djeddou, "Micro-Doppler classification for ground surveillance radar using speech recognition tools," in *Lec-*

ture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 7042 LNCS, 2011, pp. 280–287.

[44] D. Roy, S. Srivastava, A. Kusupati, P. Jain, M. Varma, and A. Arora, "One Size Does Not Fit All: Multi-Scale, Cascaded RNNs for Radar Classification," in *2019 ACM BuildSys*, 9 2019. [Online]. Available: http://arxiv.org/abs/1909.03082

[45] F. Chollet and et. al., "Keras," https://keras.io, 2015.

[46] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, and et. al., "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015. [Online]. Available: https://www.tensorflow.org/

[47] WHO, "WHO: Coronavirus disease (Covid-19) outbreak," 2020. [Online]. Available: https://www.who.int/westernpacific/emergencies/covid-19

[48] R. Rebonato, "Learning from Data," *Quantitative Finance*, vol. 16, no. 10, pp. 1479–1482, 10 2016.