Convolutional Neural Network (CNN) for Digital Radio Frequency Memory (DRFM)

by

EG Friedel

A thesis submitted to The Johns Hopkins University in conformity with the requirements for the degree of Master of Science.

Baltimore, Maryland

May, 2023

Abstract

Radar electronic jammers are evolving from hostile nations thus becoming more complex and presenting serious issues when a radar system is trying to interrogate the actual targets of interest. Electronic jammers that present a serious challenge are in a class called Digital Radio Frequency Memory (DRFM). DRFM techniques work by generating coherent false targets to the radar receiver based on an intercepted pulse signal from the radar transmitter. This will position false targets ahead or behind the radar target, thus masking the real target with false targets. The false targets can also be manipulated in amplitude, phase, and frequency.

Traditional approaches to target detection and estimation for electronic jammers generally rely on parametric modeling, which can fail because it violates the strict assumptions of classical signal processing algorithms. The result is substantial algorithm performance degradation. Furthermore, parametric models to handle DRFM jammers are difficult to design and ineffective against an evolving DRFM technology. The key to identifying opportunities for improved electronic jammer protection and

ABSTRACT

signal processing in radars is to use machine learning techniques to challenge the underlying assumptions of the standard parametric approach to design and analyze radar systems.

Convolutional Neural Networks (CNN) have gained popularity in the last few years with the advent of faster high-performance computer systems which rely on GPUs for the best computational performance. A CNN operates from a mathematical perspective and is used for non-trivial tasks such as image classification. CNNs have great performance while classifying images when they are very similar to the training dataset. However, little work has been done in developing realistic radar models which are ignoring radar environmental and antenna effects thus providing inaccurate simulation training datasets and credibility.

We propose to design a CNN that will be integrated into the radar receiver signal processing chain. The inputs to the CNN will use spatial training datasets that are the output of the radar's range-Doppler processing. The CNN will then classify DRFM false targets and feed that information to the radar detection processing component. The correct classification of DRFM targets is what we desire while processing through the radar signal returns. The predictive quality of our CNN model will drive the radar system performance and support any further actions to mitigate a DRFM jammer attack.

ABSTRACT

Primary Reader and Advisor: Dr. David Shrug

Secondary Reader: Dr. Kurt Stein

Acknowledgments

I would like to thank Dr. James Spall, Dr. David Shrug, and Dr. Kurt Stein for believing this was a worthy topic for thesis research and approving the proposal. Also, Dr. Shrug again for his encouragement and guidance throughout this challenging thesis project. Also, the Department of Applied and Computational Mathematics for allowing me to pursue this Master's degree.

I would like to thank my wife for her patience and support; while I get this done and still maintain my sanity.

Contents

Abstract **ii**

Acknowledgments **v**

List of Tables **x**

List of Figures **xi**

List of Acronyms **xiii**

1 Introduction **1**

1.1 Problem Overview 1

1.2 Background 4

1.3 Thesis Overview 7

2 Radar Concepts **8**

2.1 Overview 8

2.2 Phased Array Antenna 9

2.3 Radar Signals and Detections 10
2.3.1 Basic Radar Measurements 12
2.3.2 Radar Signal Processing 14
2.3.3 Matched Filter 15
2.3.4 Detection Fundamentals 19
2.3.4.1 Detection Hypothesis and Optimization 20
2.3.4.2 Constant False Alarm Rate (CFAR) Detection 26
2.4 Electronic Jamming Techniques 30
2.4.1 DRFM 30
2.4.1.1 No Jamming Single Target 32
2.4.1.2 Range-Bin Masking 32
2.4.1.3 Doppler-Bin Masking 33
2.4.1.4 Combined Range-Doppler Masking 34
2.4.1.5 Random Masking 34
2.4.1.6 Interference Jamming 35

3 Machine Learning **36**

3.1 Basic Concepts 36
3.2 Hyperparameters 38
3.2.1 Hyperparameters Tuning 40
3.2.2 Activation Function 41
3.2.3 Loss Function 45

3.2.4 Optimization 47
3.2.4.1 Learning Rate and Momentum 49
3.3 ALRM Method 50
3.3.1 Learning Rate Dynamic Adjustment 53
3.3.2 Momentum Dynamic Adjustment 56
3.3.3 Algorithms 57

4 Models **60**

4.1 Overview 60
4.2 Radar Simulation Model 60
4.2.1 Requirements 61
4.2.2 Data Generation 61
4.3 CNN Model 63
4.3.1 Architecture 63

5 Experiments and Results **68**

5.1 Methodology 68
5.2 Experiments 72
5.2.1 Experiment Results 72
5.3 Conclusion 76
5.3.1 Findings and Contributions 76
5.3.2 Future Work 78

6 Appendix A - Source Code **79**

6.1 MATLAB and Python Source Code 79

7 Appendix B - ResNet34 Architecture **80**

Bibliography **82**

List of Tables

4.1 Hyperparameters 65
4.2 ResNet Architectures 67

5.1 Test Accuracy and Test Loss Scores 70
5.2 Metrics 71
5.3 Experiments 72
5.4 Results Summary 77

List of Figures

1.1 Radar Processing Chain with DRFM Jammer 4

2.1 LFM Compression 12
2.2 Monopulse Antenna Quadrants 13
2.3 Standard Signal Processing Chain 15
2.4 Diagram of Two Critical Regions 25
2.5 Conventional CA-CFAR diagram from B.R. Mahafza, Radar Systems Analysis and Design using MATLAB, 2nd Ed, 2005 27
2.6 Hardware representation of an example DRFM object. 31
2.7 DRFM diagram from G. W. Stimson, Introduction to Airborne Radar, 2nd Ed, Scitech, 1998 31
2.8 Single Target Detection 32
2.9 Range-Bin Masking 33
2.10 Doppler-Bin Masking 33
2.11 Combined Range-Doppler Masking 34
2.12 Random Masking 34
2.13 Interference Jamming 35

3.1 Artificial Intelligence 37
3.2 High-Level CNN Processing Chain 39
3.3 Hyperparameters 41
3.4 ReLU Data Flow 45

4.1 CNN Feature Map Flow 63
4.2 ResNet Residual Block 66

5.1 Model Fitting 70
5.2 Experiment Measurements 72
5.3 Rosenbrock Objective Loss 73
5.4 Himmelblau Objective Loss 73
5.5 CIFAR-10 Model Accuracy 74

5.6 CIFAR-10 Loss 74
5.7 DRFM Model Accuracy 75
5.8 DRFM Model Loss 75

7.1 ResNet34 Architecture 81

List of Acronyms

Adam	Adaptive Moment Estimation
ADC	Analog-to-Digital Converter
AI	Artificial Intelligence
ALRM	Adaptive Learning Rate and Momentum
AN/TPY-2	Army Navy/Transportable Radar Surveillance
Az	Azimuth
BFGS	Broyden, Fletcher, Goldfarb, and Shannon
BMDS	Ballistic Missile Defense System
CA	Cell-Averaging
CFAR	Constant False Alarm Rate
CIFAR	Canadian Institute For Advanced Research
CNN	Convolutional Neural Network
CUT	Cell-Under-Test
DRFM	Digital Radio Frequency Memory
ECM	Electronic Countermeasures
El	Elevation
GPU	General Processing Unit
IQ	In-Phase Quadrature
IRAD	Internal Research and Development
LFM	Linear Frequency Modulation
L-BFGS	Limited-Memory BFGS
LTI	Linear Time-Invariant

ML	Machine Learning
NN	Neural Network
NP	Neyman-Pearson
NFL	No Free Lunch
PDF	Probability Density Function
ReLU	Rectified Linear Unit
ResNet	Residual Network
SGD	Stochastic Gradient Descent
SNR	Signal to Noise Ratio
SVD	Singular Value Decomposition
T/R	Transmit/Receive
WSS	Wide-Sense Stationary

Chapter 1

Introduction

1.1 Problem Overview

Ideally, radar signal processing should be able to accurately and efficiently detect false or misleading targets of interest. The reality is that typical radar signal processing is ineffective against false targets produced by deceptive electronic jammers such as DRFM. Traditional approaches to target detection and track estimation for electronic countermeasures (ECM) in general, rely on parametric modeling, that can fail because it violates the strict assumptions of classical signal processing algorithms. ECM generally attempts to interfere with or deceive the radar system with misleading electronic signals. This failure occurs because the misleading signals and the real targets are processed in the same way as the output of the analog-to-digital converter, thus the false targets and real targets are mixed together. The result is substantial

algorithm performance degradation in both target detection and track performance. The matched filter (see section 2.3.3) which optimizes the SNR for detection processing does not identify false targets. The outcome of this shortcoming is that false targets are passed through to the point of detection processing (see Figure 1.1), which then can pollute the tracker and further task scheduling of radar resources towards the misleading signals. We argue that if we can identify the misleading signals, then the radar can take steps to mitigate or cancel the ECM. We also propose to identify that we have misleading signals before processing the detections, thus saving the radar from unnecessary computing cycles to process the false targets and prevent overall algorithm performance degradation. Chapter 2 provides a discussion on radar concepts that hopes to make clear from a mathematics perspective and an explanation that there are no clear rules on target deception by misleading signals.

We propose to use machine learning techniques to challenge the underlying assumptions of the standard parametric approach for the design and analysis of radar systems. Convolutional Neural Networks (CNN) have gained popularity in the last few years with the advent of faster high-performance Graphics Processing Unit (GPU) computers. Current research demonstrates CNN as a sound approach for radar signal classification with more work to be done. We will show a Convolutional Neural Network (CNN) that will use spatial training datasets in the form of range-Doppler images to perform radar signal classification. The radar signal processing will then be examined to identify different ECM classes. The goal is to show that ECM can be

mitigated and thus improve overall radar performance operation and target recognition. We focus on a particular type of active ECM called Digital Radio Frequency Memory (DRFM). We choose to focus on the DRFM jammer because it's the most problematic ECM for radars to date.

Success is defined by the identification of an ECM, in this case, a DRFM jammer, and its type via the CNN model. The new CNN model is proposed to be part of the radar's signal processing chain. The DRFM jammer will fail to affect radar detection and tracking operations in the radar. Identification is the first step in mitigating the effects of ECM.

The status quo is the failure to identify false targets or interference is what we face today since DRFM is problematic for typical signal processing. The failure effects are as follows:

- Impact on system performance due to more signal processing operations
- Degrades detection performance
- Hides the real targets, deceiving the Tracker

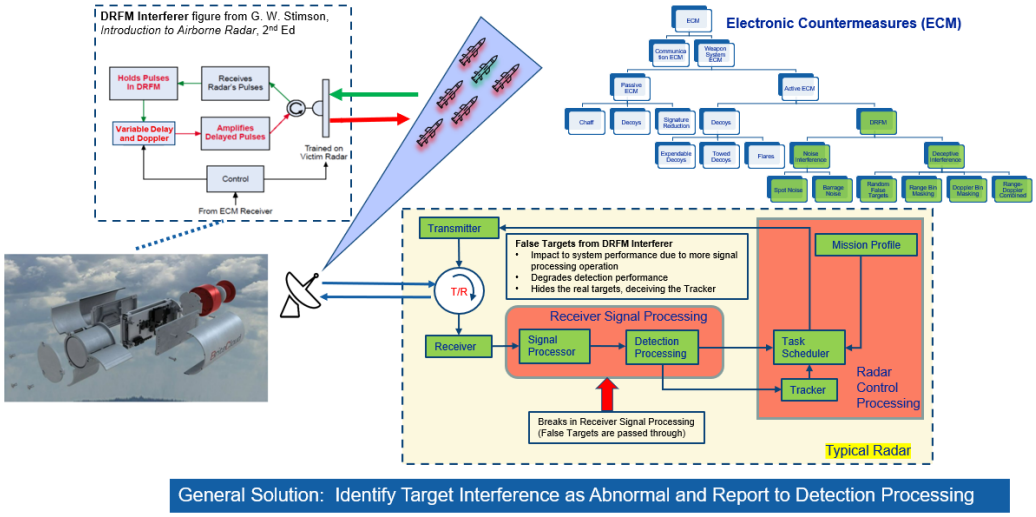


Figure 1.1: Radar Processing Chain with DRFM Jammer

This thesis proposes a novel CNN design exploring gradient descent approaches and cross-entropy loss functions in the application of image classification of radar targets. We put this in the category of machine learning. We cast this as an optimization problem where we are attempting to minimize (θ^*) over some loss function $L(\theta)$ to specifically solve an image classification problem.

1.2 Background

In recent years, the development of radar jamming technology is advancing and becoming more problematic for radar signal processing. Digital Radio Frequency Memory (DRFM) is one of the emerging technologies in the area of active deceptive jamming that is becoming problematic for existing radar capability to keep pace. [1].

An active radar jamming system using DRFM as the central subsystem in the generation and detection of specific signal characteristics was patented by Richard Wiegand in 1985. [2] DRFM works by intercepting the transmitting radar's pulse energy and then storing in its device memory the pulse energy characteristics, thus attempting to match the transmitted pulse. The matching of the transmitted pulse is troublesome because typical signal processing uses a matched filter approach to separate the detection signal from the noise which is dependent on the transmitted pulse. The DRFM will then radiate a signal with a time or Doppler delay based on the stored pulse energy characteristics, which will present a false target to the receiving radar. The false target is directed towards the radar main-lobe which makes this a hard problem since this will get processed as a real target detection. We generally define this as deceptive jamming.

Deceptive jamming is a very active field of study in the radar community within the last decade. Major efforts focus on classifying the jamming signal, where our interest lies. Various techniques have been studied such as the use of a likelihood generalized function to determine a false target produced by the jammer and a real target of interest. [3] [4] Likelihood methods are used in typical signal detection processing and rely on apriori information to match the jammer. This approach falls short in the real world of jammer evolution.

More recently, machine learning techniques have been proposed which have founded

success in image recognition. [5] Machine learning was first conceived from the mathematical modeling of neural networks, which was published in 1943 by Walter Pitts and Warren McCulloch. Through the decades various applications and advancements have occurred. Most notably in 2006 with "deep learning" where Geoffrey Hinton developed algorithms that helped computers recognize different types of objects and text characters in pictures and videos. Also in 2010, ImageNet classification was conceived which dramatically improved the accuracy of machine learning and artificial intelligence image recognition. [6]In terms of radar jammer classification research since 2018 we note some of the relevant contributions which are as follows: A dense false target jamming recognition algorithm using a special case of the short-time Fourier transform called a Gabor transform is proposed. [7]. Another idea is to use a decision tree algorithm approach to recognize jamming. [8] A more traditional approach is a feature fusion algorithm based on Bayesian decision theory applied to the recognition of radar deception jamming. These algorithms seem to rely on subject matter expert knowledge in terms of feature extraction and judgment. They do also appear to be ambiguous in areas and can lead to subjectivity, thus leading to problems in robustness and accuracy. There are other contributions [9] [10] [11] that seem well suited but require a large number of computation resources because of the complexity of the feature extractions. This thesis will focus on image recognition and hopes to address some of the shortcomings in the past and currently active research to make contributions in both the radar and machine learning domains.

1.3 Thesis Overview

The remainder of the thesis is as follows:

- Chapter 2 - A review of radar concepts that are essential to the test data generation and the problem space. We will consider a ground-based phased array radar system and how we extract information from the radar signals which are then used to train our neural network. We also try to establish that there are no clear rules or real precedents for deceptive target identification or ECM migration in typical radar signal processing by examining the mathematics.

- Chapter 3 - A review of machine learning concepts. An introduction to the theory and optimization techniques choices for the CNN. We will focus specifically on the most important aspects of a CNN and justify how that applies to our problem space to then propose our algorithms and solution approach.

- Chapter 4 - The Radar Model requirements and data generation will be presented, along with the CNN Model architecture approach.

- Chapter 5 - The methodology for the experiment testing will be presented with the metrics and the data analysis results. The findings and conclusions of the thesis are presented with the proposed future work.

Chapter 2

Radar Concepts

2.1 Overview

This chapter provides some basic radar concepts that help to support this thesis. We will consider for this thesis a phased array radar system and describe its basic signal processing capabilities and techniques. We will also introduce electronic jammers and the various techniques fundamental to this thesis's groundwork. Most of the content is based on years of work-related experience in this field.

The chapter content is mostly supported by standard references such as Skolnik and Richards that are well-known and established throughout the radar community.

2.2 Phased Array Antenna

When we think of radar it's common to picture the antenna in our minds. The antenna selection is a unique and important part of any radar. There are various types of antennas used in radar applications. This thesis will incorporate and model a ground-based phased array radar system. We focus on using the phased array antenna since this is an antenna type used for the Ballistic Missile Defense System (BMDS) [12] in defense of the United States against ballistic missile threats from hostile nations. A phased array antenna system is typically made up of many transmit and receive (T/R) elements. The T/R element is typically housed within the same module and configured in some geometrical pattern. The AN/TPY-2 radar, for example, has over 25,000 T/R modules arranged in a rectangular pattern that has an array size of about 9.2 m^2. The phased array steers the direction of the beams by electronic time-delay or phase shifting of the transmit/receive (T/R) elements. There is no mechanical motion to direct the antenna toward its intended target of interest. Another feature of an antenna system to consider is how the beams are formed. Analog and digital are the two general approaches to beamforming.

Analog beamforming relies on a single electromagnetic signal as the input to each T/R module. This signal can be phase shifted and amplified which is done before the analog-to-digital converter (ADC). This has been the approach for the BMDS radars and is considered a cost-effective way of building phased array systems. It can only generate a single beam at a time and thus does not adapt very well to a changing

environment.

Digital beamforming converts the electromagnetic signal at each T/R module into two streams of digital data that are 90 degrees from each other in terms of sin and cos functions. This gives a complex sample at each element of the array. This allows for a digital signal earlier in the signal processing chain. Digital radars are quickly becoming more popular and cost-effective since it allows the radar to not only adapt to various array configurations but provides more timely digital data and thus is part of the motivation for this thesis.

2.3 Radar Signals and Detections

Radar signals in the form of electromagnetic energy are both transmitted and received respectively by the transmitter and receiver devices of a radar. A radar typically transmits a waveform modeled by the complex function

$$\bar{s}(t) = A(t)e^{i2\pi f_0(t)+\theta(t)} \tag{2.1}$$

where $A(t)$ is the amplitude over time, f_0 is the carrier frequency of the pulsed waveform, and $\theta(t)$ is the phase component from $[0, \pi]$. This is also considered to be the complex envelope of a linear frequency-modulated (LFM) waveform. The transmitted waveform or commonly known as a radar beam can be represented by a

rectangular envelope with a constant amplitude $A(t)$ where τ is the pulse duration.

$$A(t) = \begin{cases} 1 & : \ 0 \leq t \leq \tau \\ 0 & : \text{otherwise} \end{cases} \tag{2.2}$$

The LFM waveform which is Doppler tolerant and common for this type of radar will produce a linear sweep across a defined bandwidth β for a given pulse-width τ defined by

$$x(t) = cos\left(\pi \frac{\beta}{\tau} t^2\right) \quad 0 \leq t \leq \tau \tag{2.3}$$

The linear sweep in frequency naturally compresses the signal when processed which makes for excellent Doppler response, which is desired when using range-Doppler images. We do however point out that a large Doppler shift can produce undesired range-Doppler coupling that shifts the target falsely in range due to the increased target velocity. It will produce range and Doppler smearing (see Figure 2.5 Range-Bin Masking), which we see in some of the generated range-Doppler images used in the CNN. We include this because it's a real-world effect. The plot below is from our radar MATLAB model and is using a representative set of parameters that generates our range-Doppler images that demonstrate the desired pulse compression.

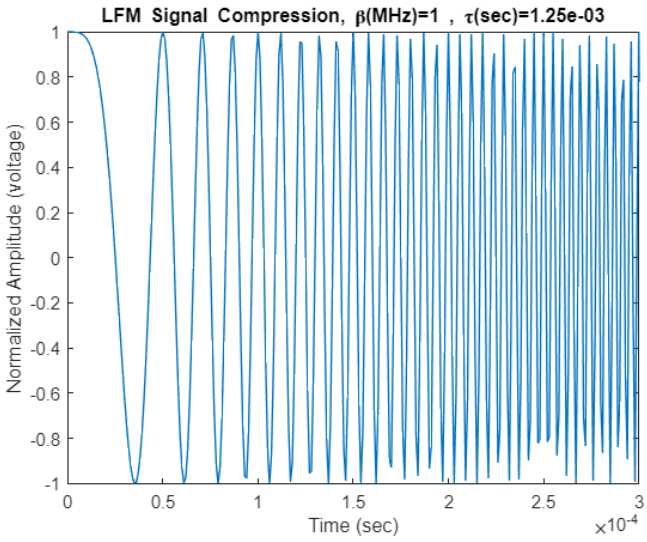


Figure 2.1: LFM Compression

2.3.1 Basic Radar Measurements

We start with a dataset that contains digital samples called In-phase and Quadrature (IQ) data. The IQ is the result of converting an analog signal from the radar-received signals to a complex number via an analog-to-digital converter processing computer chip. We assume an N-samples data set $x[0], x[1], ...x[N-1]$. The IQ for each sample in the data set is then a complex number defined by

$$e^{i2\pi ft} = cos(2\pi ft) + isin(2\pi ft) \tag{2.4}$$

where f is the frequency in Hz, $cos(2\pi ft)$ represents the magnitude of the vector, in-phase I component and $i\sin(2\pi ft)$ represents the phase, quadrature Q component.

We consider the IQ as a rotating vector in the complex plane. Radar is a five-dimensional sensor and these dimensions are the origin of radar signal information that can be derived from the IQ data and the setup of the radar array. The five dimensions are as follows in no particular order: Doppler, range, azimuth, elevation, and amplitude.

$$\textit{Target Doppler} \qquad f_d = \frac{2v}{c} f_c = \frac{2v}{\lambda_t} \tag{2.5}$$

$$\textit{Target Range} \qquad R = \frac{c\Delta t}{2} \tag{2.6}$$

$$\textit{Target Azimuth} \qquad \theta_{Az} = \Delta_{Az}/\Sigma \tag{2.7}$$

$$\textit{Target Elevation} \qquad \theta_{El} = \Delta_{El}/\Sigma \tag{2.8}$$

$$\textit{Target Amplitude} \qquad A_{dB} = 10 * log_{10}\left[\frac{I^2 + Q^2}{V_{FullScale}}\right] \tag{2.9}$$

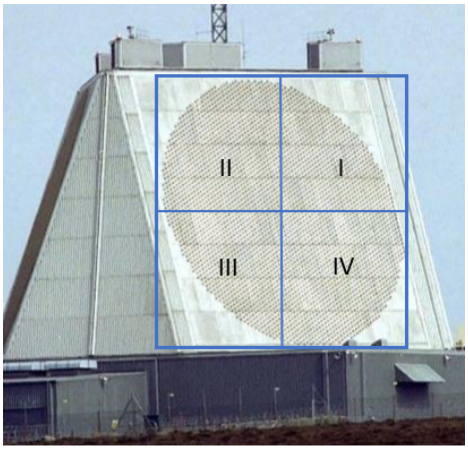


Figure 2.2: Monopulse Antenna Quadrants

The delta (Δ_{Az}, Δ_{El}) and summation ($\sum$) channels can be expressed by either the

amplitude or phase measurements done at the antenna. Segments I, II, III, and IV split the antenna array face up to allow for monopulse measurements. For example, a target that impinges the radar at the origin (i.e. boresight) would calculate a difference of zero in the azimuth and elevation channels, since the amplitude or the phase is centered on the quadrant grid. This is considered a monopulse measurement since it only requires a single pulse to determine the target angle and signal strength.

$$\sum = (I + II + III + IV) \tag{2.10}$$

$$\Delta_{Az} = (I + IV) - (II + III) \tag{2.11}$$

$$\Delta_{El} = (I + II) - (III + IV) \tag{2.12}$$

2.3.2 Radar Signal Processing

Standard radar signal processing starts with and uses a generalized matched filter receiver. The input for the matched filter is the IQ data samples discussed in the previous section. We do note that the IQ gets filtered down to a baseband signal thus removing the carrier frequency and leaving the original baseband signal. The range-Doppler maps are dependent on the matched filter output and serve as the primary inputs to our CNN. We will show how the matched filter transforms the output data. The figure below provides the primary components of a standard radar signal processing approach and where we will apply the CNN for image classification.

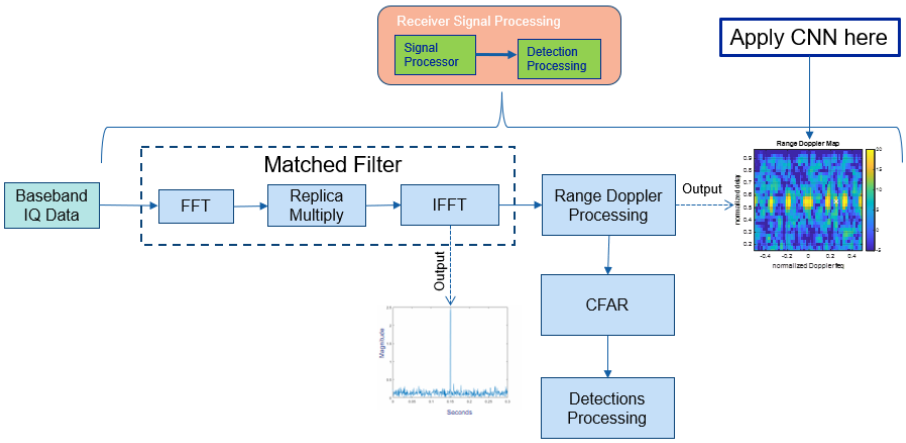


Figure 2.3: Standard Signal Processing Chain

2.3.3 Matched Filter

The matched filter is ubiquitous in radar signal processing and essential for detection processing, thus providing an essential signal processing step in the creation of the range-Doppler images. There is much documented about the matched filter, we cite Skolnik [13] as the reference which is preferred within the radar community. The matched filter attempts to optimize the signal-to-noise (SNR) of a detection in Gaussian noise. The replica or copy of the transmitted signal, in our case the LFM waveform, is used as a time-reversed and complex conjugate version of the transmitted signal to form the matched filter, which is then convolved with the received signal. We now provide the matched filter derivation to give mathematical reasoning to the range and frequency response of the received signal which makes up the range-Doppler images that are used as inputs to our CNN. We can express a matched filter

in various forms as the Fourier transform of the finite received signal $s(t)$ at some time t, where we define a measured signal with noise as $s_i(t) = x_i(t) + n_i(t)$. Where i depends on the number of samples. The noise $n_i(t)$ is defined as a random process where its mean and auto-correlation functions are time-invariant, thus the function $n_i(t)$ is defined as wide-sense stationary (WSS). Now we consider an impulse response $h(t)$ with a Linear Time-Invariant (LTI) filter that takes our measured signal $s_i(t)$ and outputs an optimal filtered response $s_o(t) = x_o(t) + n_o(t)$. We now develop the equations for maximizing the signal-to-nois99e ratio (SNR). We first define the signal $s_o(t)$ and noise $n_o(t)$ outputs separately since we are interested in maximizing the ratio of these terms.

$$n_0(t) = n_i(t) * h(t) = \int_{-\infty}^{\infty} n_i(u)h(t-u)du \tag{2.13}$$

$$s_0(t) = x_i(t) * h(t) = \int_{-\infty}^{\infty} x_i(u-t_0)h(t-u)du \tag{2.14}$$

Now we need to express these equations in terms of signal and noise power. We assume an auto-correlation function for noise power since the input noise is band-limited and WSS. So then we write

$$N_{n_i(t)} = \left(\frac{\eta_0}{2}\right)\delta(t) \tag{2.15}$$

where $\left(\frac{\eta_0}{2}\right)$ is the constant noise source and $\delta(t)$ is the impulse signal commonly referred to as the Dirac delta function. The auto-correlation in terms of the Fourier

transform of the output noise is

$$N_{n_o(t)} = \left(\frac{\eta_0}{2}\right) |H(f)|^2 \tag{2.16}$$

now expressing as the noise power we get

$$\sigma_{n_0}^2 = \left(\frac{\eta_0}{2}\right) \int_{-\infty}^{\infty} |H(f)|^2 \, df. \tag{2.17}$$

For the signal power, we assume a normalized signal power at the output of the matched filter Using Parseval's theorem [14] the total energy of some signal E_x is

$$E_{x_o} = |x_o(t)|^2. \tag{2.18}$$

Taking Eq. (2.14) and applying a Fourier transform we get

$$S_0(f) = X_i(f) H(f) e^{i2\pi f t_0} \tag{2.19}$$

Integrating

$$s_0(t_0) = \int_{-\infty}^{\infty} X_i(f) H(f) e^{i2\pi f t_0} \, df \tag{2.20}$$

Now apply Eq. (2.18) we get the signal power as

$$E_{x_o} = \left| \int_{-\infty}^{\infty} X_i(f) H(f) e^{i2\pi f t_0} \, df \right|^2 \tag{2.21}$$

Combing the signal and the noise as a ratio we get

$$SNR(t_0) = \frac{\left| \int_{-\infty}^{\infty} X_i(f) H(f) e^{i2\pi f t_0} \, df \right|^2}{\left(\frac{\eta_0}{2}\right) \int_{-\infty}^{\infty} |H(f)|^2 \, df} \tag{2.22}$$

The question to ask is what receiver frequency response $H(f)$ will maximize the SNR. We assert that the optimum SNR at the output of the receiver-matched filter is expressed in Eq. (2.22) when the numerator is at its maximum. To prove this we can use the Cauchy-Schwarz inequality to establish that we have a maximized SNR dependent on the receiver frequency response $H(f)$. Applying the Cauchy-Schwartz inequality to Eq. (2.22) gives

$$SNR(t_0) = \frac{\left| \int_{-\infty}^{\infty} X_i(f) H(f) e^{i2\pi f t_0} \, df \right|^2}{\left(\frac{\eta_0}{2}\right) \int_{-\infty}^{\infty} |H(f)|^2 \, df} \leq \frac{\int_{-\infty}^{\infty} |X_i(f)|^2 \, df \;\; |H(f)|^2 \, df}{\left(\frac{\eta_0}{2}\right) \int_{-\infty}^{\infty} |H(f)|^2 \, df} \tag{2.23}$$

$$\leq \frac{2 \int_{-\infty}^{\infty} |X_i(f)|^2 \, df}{\eta_0} \tag{2.24}$$

Equality is assured if and only if $H(f) = \alpha X_i^*(f)$[1], where $X_i^*(f) = H(f) e^{i2\pi f t}$ and α

[1]Starred transform which denotes a discrete-time version of the Laplace transform

is some arbitrary constant. Thus the SNR is maximized when

$$h(f) = \alpha x_i^*(t_0 - t) \tag{2.25}$$

$$H(f) = \alpha X_i^*(f) e^{-i2\pi f t_0} \tag{2.26}$$

Thus we can express the final SNR_{max} equation as

$$SNR_{max}(t_0) = \frac{2E_{x_i}}{\eta_0} \tag{2.27}$$

2.3.4 Detection Fundamentals

The detection itself is fairly straightforward, in which the received signal power with noise is compared to some threshold value. A detection is positively identified if the received signal power exceeds the threshold value. Thus the threshold value is critical to establishing detections. In general, the threshold value is a function of both the probability of false alarms and the probability of detections. A target is detected based on establishing a threshold at the output of the radar receiver. If the receiver output is large enough to exceed the threshold, a target is said to be present. The process also then must decide if this detection measurement represents the influence of a target or only signal interference. Statistical signal models are typically used. This then becomes a problem of statistical hypothesis testing.

A primary function of any radar system is the function of generating and process-

ing signal detections. The detection itself is binary, meaning it can be a real or false target of interest. A real target is the signal energy received by the radar receiver reflected from a real target of opportunity. A false target can be interference, random noise, or a jammer that is radiating noise or some deceptive signal toward the radar receiver. If it's decided that we have a real target then further detection processing usually happens. A problem occurs when a false target can not determine and thus is considered a real target. The radar can be making up to a million detection decisions per second. Since we are dealing with a large amount of data for detection decisions, typical radar targets are represented by statistical signal models. This has limitations since it assumes that the noise and interference levels are known. It also does not account for false targets that are deceptive, which thus appear as real targets when statistical hypothesis testing is applied.

The digital radars of today produce the range-Doppler data that we use to generate our Range-Doppler images, thus we are not adding any additional processing or data constraints to the radar system. The range-Doppler data is the input for our CNN that will be used to train our models that is then used to identify radar false targets.

2.3.4.1 Detection Hypothesis and Optimization

The standard approach for target detection for any radar measurement depends on two hypotheses, in which one of the two hypotheses is assumed to be true. The radar measurement consists of either

1. The radar measurement contains only noise energy. No target is present.

2. The radar measurement is a combination of noise and target return energy. A target is present.

We regard this as a binary classification problem. Since we describe the signals statistically we consider the following probability density function (pdf) that describe the radar measurements to be tested under our two hypothesis (1.,2. from above).

$$p_x(\mathbf{x}|H_0) = \text{pdf of } \mathbf{x} \text{ given that the target was not present}$$

$$p_x(\mathbf{x}|H_1) = \text{pdf of } \mathbf{x} \text{ given that the target was present}$$

where $\mathbf{x}$ is a column vector of N samples based on some x_n data such that

$$\mathbf{x} \equiv [x_0, x_1, x_2, ...x_{N-1}]'$$

This gives N-dimensional joint pdfs $p_x(x|H_0)$ and $p_x(x|H_1)$, used to consider the following probabilities for use cases which are figures of merit when describing detection performance.

1. Probability of Detection (P_D): A target is present and detected.

2. Probability of False Alarm (P_{FA}): No target is present but a detection is identified.

3. Probability of a Missed Detection (P_M): A target is present but there is no detection identified where $P_M = 1 - P_D$.

The probability of detection and the probability of false alarm is of interest and can be expressed as integrals of joint probabilities over some region $\mathcal{R} = x : P(H_1|x) > l_0$, where l_0 is a threshold defined by $0 \leq l_0 \leq 1$.

$$P_D = \int_{\mathcal{R}} p_x(\mathbf{x}|H_1)dx \tag{2.28}$$

$$P_{FA} = \int_{\mathcal{R}} p_x(\mathbf{x}|H_0)dx \tag{2.29}$$

Based on the detection hypothesis we need a method to optimize our choice. Most radar detection processing uses a Bayes optimization criterion that assigns a cost to a target being present or not present. In most radar systems a special case of the Bayes theorem called the Neyman-Pearson (NP) detection criterion is the standard approach. [15] We now will develop the mathematics behind the NP theorem developing a variation to the Gibra explanation. [16]

Assume that the observation of a random variable whose probability density function (pdf) is $\mathcal{N}(\mu, \sigma^2)$ which is a Gaussian pdf with a mean μ and variance σ^2. We now must determine if the mean is $\mu = 0$, (target not present: $\mathcal{N}[0, 1]$) or $\mu = 1$,

(target present: $\mathcal{N}[1,1]$). Now we can think of the μ as a hypothesis test such that

$$\mathcal{H}_0 : \mu = 0 \qquad \textit{(null hypothesis)} \tag{2.30}$$

$$\mathcal{H}_1 : \mu = 1 \qquad \textit{(alternative hypothesis)} \tag{2.31}$$

Now the problem is reduced to a binary hypothesis problem where we choose between the null or alternative hypothesis.

Theorem 1. *(**Neyman-Pearson**) Given some probability of false alarm $P_{fa} = \gamma$ determine if a target is present $\mathcal{H}_1$ using a likelihood ratio function $L(\boldsymbol{x})$ if*

$$L(\boldsymbol{x}) = \frac{p(\boldsymbol{x} : \mathcal{H}_1)}{p(\boldsymbol{x} : \mathcal{H}_0)} > \gamma \tag{2.32}$$

where the threshold α is determined by

$$P_{fa} = \int_{x:L(x)>\alpha} p(\boldsymbol{x} : \mathcal{H}_\prime)\, dx = \gamma \tag{2.33}$$

Proof. Let R be a critical region determined by the NP criteria, where the critical region given by Eq.(2.30) minimizes the error of Type II (false negative conclusion) for a large range of alternatives. Now suppose that the critical region R is not the best choice for Type II error minimization. Now let R' be the better choice for Type II error minimization. We assume that both regions R, R' are of the same size α. We

also introduce a probability of detection P_d where

$$P_d = (1 - P_{fa}) \tag{2.34}$$

We now approach the proof assuming a minimized P_{fa} that provides a maximum P_d so we proceed in that way. We can use Lagrangian multipliers to maximize the P_d for some given P_{fa}. Using the Lagrangian we get

$$F = P_d + \lambda(P_{fa} - \gamma) \tag{2.35}$$

$$= \int_{R'} p(\mathbf{x} : \mathcal{H}_1)dx + \lambda \left(\int_{R'} p(\mathbf{x} : \mathcal{H}_0)dx - \gamma \right) \tag{2.36}$$

$$= \int_{R'} \left[p(\mathbf{x} : \mathcal{H}_1) + \lambda p(\mathbf{x} : \mathcal{H}_0) \right] dx - \lambda\gamma \tag{2.37}$$

Now we consider two cases

$$\textit{case 1:} \quad \int_{R'} \left[p(\mathbf{x} : \mathcal{H}_1) + \lambda p(\mathbf{x} : \mathcal{H}_0) \right] dx > 0 \tag{2.38}$$

$$\textit{case 2:} \quad \int_{R'} \left[p(\mathbf{x} : \mathcal{H}_1) + \lambda p(\mathbf{x} : \mathcal{H}_0) \right] dx = 0 \tag{2.39}$$

For *Case 1* to maximize F the integral is positive and $\mathbf{x}$ is included in the region R'. For *Case 2* when the integral $\int_{R'}(0)\ dx = 0$, $\mathbf{x}$ can be found in both regions of R and R'.

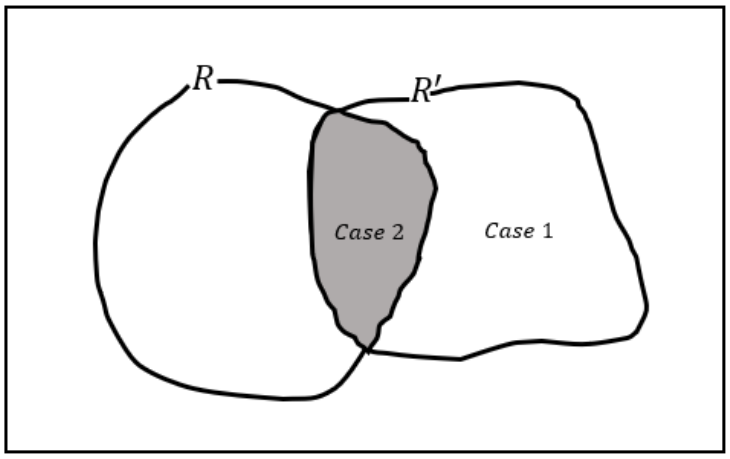


Figure 2.4: Diagram of Two Critical Regions

We seek to determine the alternative hypothesis and if a target is present we state that

$$\frac{p(\mathbf{x} : \mathcal{H}_1)}{p(\mathbf{x} : \mathcal{H}_0)} > -\lambda \tag{2.40}$$

Given that the likelihood ratio is always positive and assuming that the PDFs are continuous *Case 2* is not required. We now see that $\lambda < 0$ must be true otherwise the likelihood ratio exceeds a negative value. So then we let $\alpha = -\lambda$ which gives

$$\frac{p(\mathbf{x} : \mathcal{H}_1)}{p(\mathbf{x} : \mathcal{H}_0)} > \alpha \tag{2.41}$$

where the threshold value $\alpha > 0$ is determined from the $P_f a = \gamma$. □

2.3.4.2 Constant False Alarm Rate (CFAR) Detection

An approach for typical threshold detection processing was discussed in Detection Fundamentals, we now expand on this to include an approach where the noise and interference levels can be variable which is often the case with a real-world radar application. We desire an approach that will adapt to the changing interference and noise environment to establish a defined false alarm rate parameter over some given probability of detection. This concept is fundamental to the radar system's ability to maintain an acceptance false target threshold level. We desire to maintain a constant predictable false alarm rate, the detection threshold will increase or decrease in proportion to the noise power in the reference cells. The probability of a false alarm is independent of the noise power. The probability distribution in the detection of signals with noise follows a Rayleigh distribution. There are various forms of CFAR we will focus on what works best against typical noise jammers. Cell-Averaging (CA)-CFAR is proven to be superior [17]**f**or a type of noise or target jamming and is the most widely used. This is considered an adaptive detection threshold technique that assumes that the probability density function of the noise is known. The CA-CFAR process flow is depicted in Figure 2.3. The output of the range-Doppler processing is the input to the square law detector (see Figure 2.2). Also worth noting is that this same input is also used for our CNN. We will argue that the CNN model approach will identify false targets more readily thus offering a better solution than the conventional CA-CFAR approach.

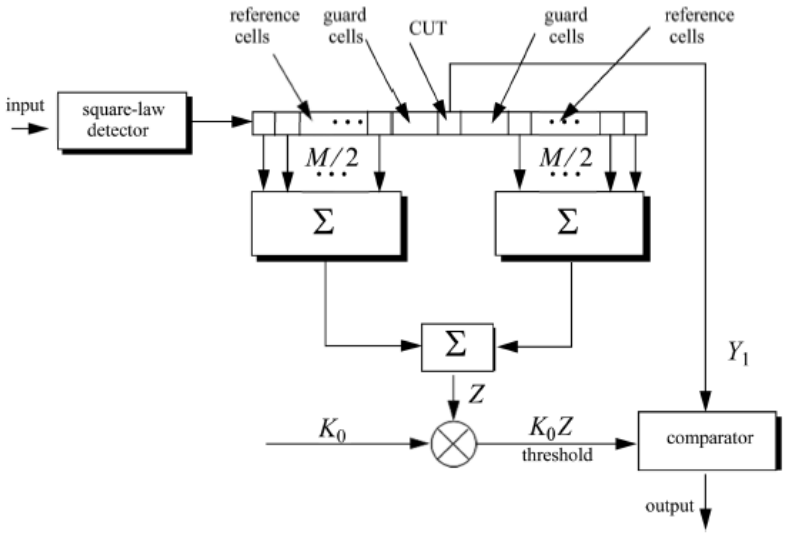


Figure 2.5: Conventional CA-CFAR diagram from B.R. Mahafza, Radar Systems Analysis and Design using MATLAB, 2nd Ed, 2005

The CUT is the Cell-Under-Test. The guard cells provide support to inhibit the signal from leaking into the reference cells and distorting the noise estimate. The reference cells are made up of range and Doppler complex sample data which is also used to produce the range-Doppler images. In general, the number of reference cells before and after the CUT is symmetric. We now will develop the (CA)-CFAR mathematics which closely follows Figure 2.3.

We start with a definition of the square-law detector threshold.

$$T_{sqLaw} = K_0 Z \tag{2.42}$$

where the noise power estimate Z and scalar threshold K_0 is

$$Z = \sigma^2 = \frac{1}{M}\sum_{t=1}^{M} s_t \tag{2.43}$$

$$K_0 = -ln(P_{fa}) \tag{2.44}$$

While we did provide an equation for P_{fa} in Eq. (2.29) we should expand more on the mathematical details to build a better intuition for the (CA)-CFAR. We consider a conditional P_{fa} which we average over all the K_0 threshold values such that

$$P_{fa}(K_0) = e^{-K_0/Z} \tag{2.45}$$

We desire a P_{fa} that is independent of the noise power which is the overall objective of conventional CA-CFAR processing. So then, we define an unconditional probability as

$$P_{fa} = \int_0^\infty P_{fa}(K_0) f(z) dz \tag{2.46}$$

We then declare that all the reference cells are Gaussian and not skewed (i.e. zero mean) with a σ^2 variance. Our noise estimate Z is considered a random variable with

a gamma probability distribution function where

$$f(z) = \frac{z^{(\frac{M}{2}-1)}e^{(\frac{-z}{2Z})}}{2^{\frac{M}{2}}Z^{M}\Gamma(\frac{M}{2})}), \quad z > 0. \tag{2.47}$$

The function $f(z)$ is the pdf of the threshold which follows Eq. (2.47) above, which now we define in terms of our scalar threshold K_0, therefore,

$$f(z) = \frac{z^{(M-1)}e^{(\frac{-z}{2K_0Z})}}{(2K_0Z)^{M}\Gamma(M)}, \quad z \geq 0. \tag{2.48}$$

Substituting $f(z)$ into Eq. (2.46) and integrating we get the P_{fa} independent of the noise power.

$$P_{fa} = \frac{1}{(1+K_0)^{M}} \tag{2.49}$$

so then a detection is declared for the CUT at the output of the comparator when

$$Y_1 \geq T_{sqLaw}. \tag{2.50}$$

Note that this is for a single pulse, a similar process can be followed for multiple pulses.

2.4 Electronic Jamming Techniques

Electronic jamming covers various techniques and spans many areas. We will focus on the area of *Deceptive Jamming*. We will look at a class of electronic jammers called smart jammers, commonly referred to as deceptive jammers or DRFM jammers. Jammers in general are specifically employed to radiate interference via noise or signal towards the opposing radar thus jamming the opposing radar receiver. There are two types of methods for electronic jammers. They are noise and repeaters. Today's electronic countermeasure (ECM) equipment can work both as a deceptive and a noise jammer, while in the past these were two different classes of equipment. Any ECM aims to reduce the operational capability of a system. In our case, the system is a ground-based phased array radar.

2.4.1 DRFM

The DRFM can intercept transmitting radar signals (see section 2.3 Radar Signals) through high-speed digital sampling and then stores the samples in memory for later use. DRFM then uses a repeater technique that can modify and re-transmit the stored transmitted signal by adjusting the time or frequency. This results in a false target that is radiated to the opposing radar receiver thus deceiving the radar with a false target position. The deception or false target occurs because the radar echo radiation is matched to the opposing radar's receiver, thus resulting in a false target

detection. A DRFM-based repeater stores each received pulse enabling precise range and Doppler walk-off. The DRFM can represent multiple false targets at once thus making it very problematic for typical radar detection processing. The DRFM can be launched from a decoy dispenser or towed behind a missile threat since it is small (55 mm diameter) and light-weight (1.1 kg) [18]

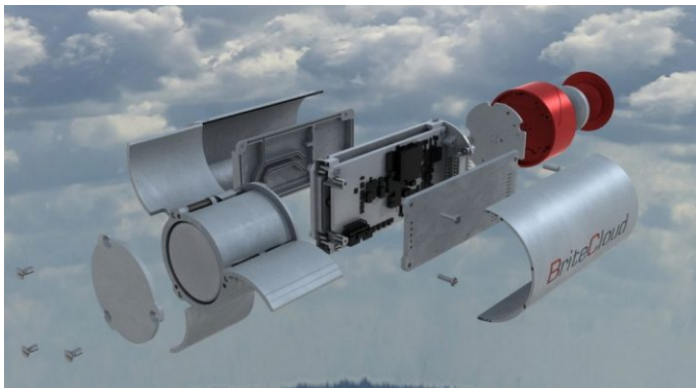


Figure 2.6: Hardware representation of an example DRFM object.

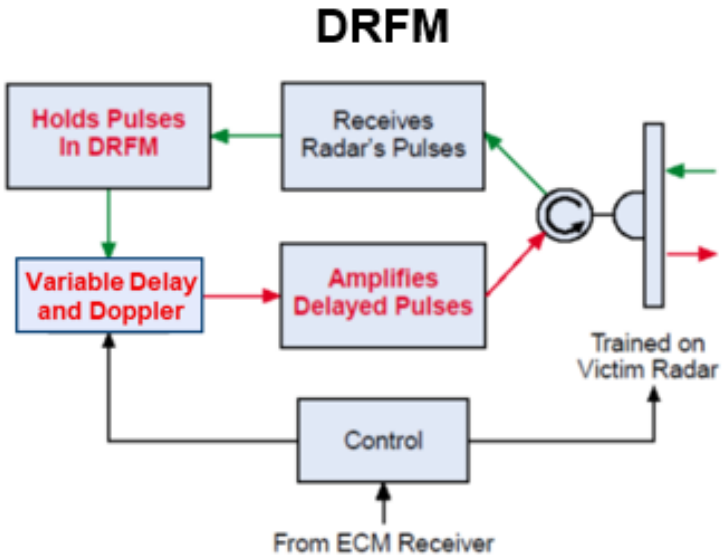


Figure 2.7: DRFM diagram from G. W. Stimson, Introduction to Airborne Radar, 2nd Ed, Scitech, 1998

We now will expand on the different types of DRFM jamming that will become the basis for our range-Doppler images and classes in our CNN. The example figures that follow are generated by our MATLAB Radar model.

2.4.1.1 No Jamming Single Target

We first consider the case where no jamming is present and there is a truly intended target detection. We mustn't miss-classify a valid detection.

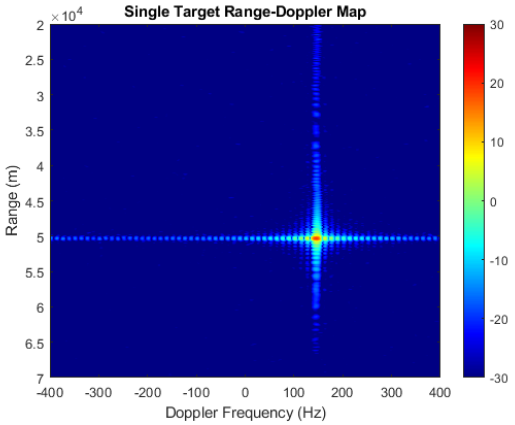


Figure 2.8: Single Target Detection

2.4.1.2 Range-Bin Masking

Range-bin masking obscures the target range by introducing a time-delayed Δt return signal back to the radar receiver which results in various false target detections lined up in the range dimension and centered about the true intended target. Recall that target range is defined by $R = \frac{c\Delta t}{2}$

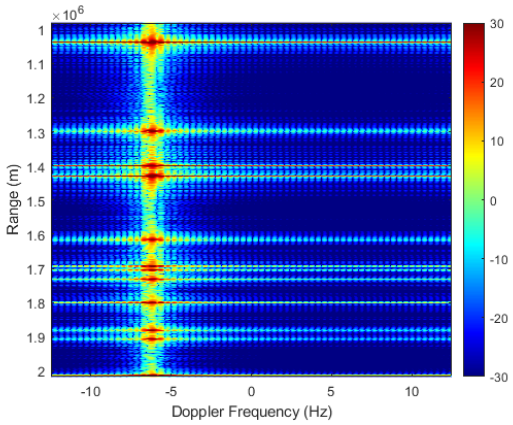


Figure 2.9: Range-Bin Masking

2.4.1.3 Doppler-Bin Masking

Doppler-bin masking obscures the target Doppler response by introducing a variety of slow-time phase modulated signals back to the radar receiver which affects the target velocity v and results in various false target detections lined up in the Doppler dimension and centered about the true intended target. Recall that $f_d = \frac{2v}{c} f_c = \frac{2v}{\lambda_t}$.

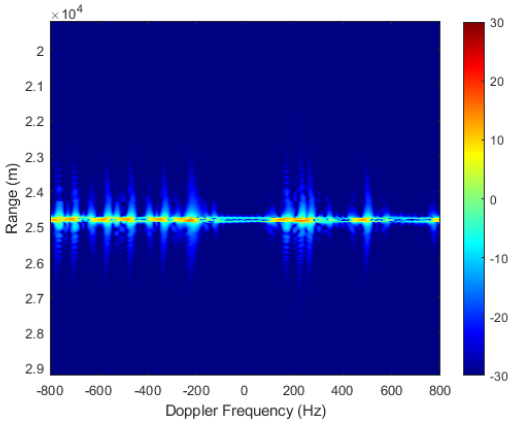


Figure 2.10: Doppler-Bin Masking

2.4.1.4 Combined Range-Doppler Masking

Multiple range and Doppler bins can be sent by a DRFM device at once.

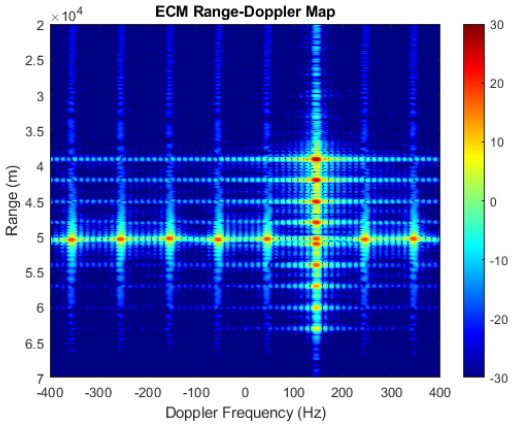


Figure 2.11: Combined Range-Doppler Masking

2.4.1.5 Random Masking

We also consider random false targets sent by a DRFM device.

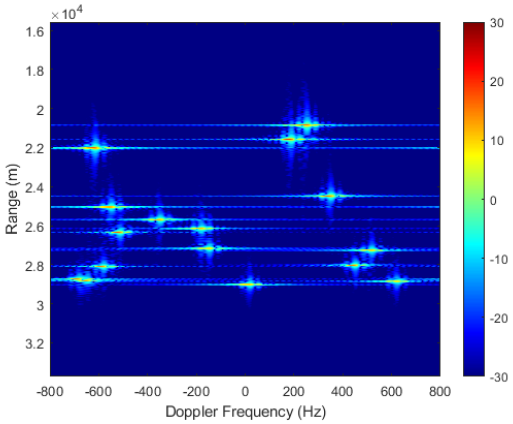


Figure 2.12: Random Masking

2.4.1.6 Interference Jamming

This DRFM will effectively hide the target in range for this example. This is considered interference since it appears as vertical lines in the range-Doppler image.

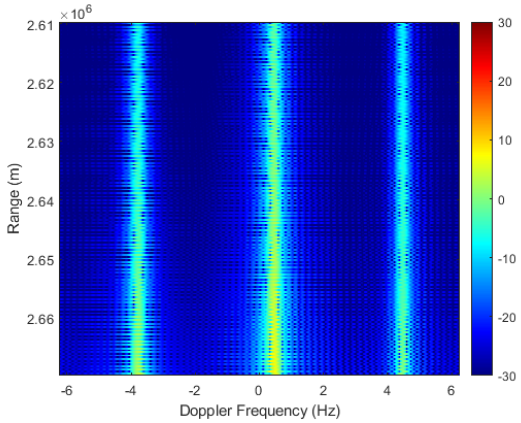


Figure 2.13: Interference Jamming

Chapter 3

Machine Learning

3.1 Basic Concepts

We are still in the age of information with the advent of bigger and faster computers we have been able to store and process more and more big data. An understanding of basic concepts and terminology of machine learning will be addressed here and thus provides a foundation for the material that follows. Arthur Samuel in 1959 popularized the term "Machine Learning" (ML) which is the field of study that gives computers the ability to learn without explicitly being programmed. Since ML is considered an algorithm technique this falls under the larger category of Artificial Intelligence (AI). We define AI as any technique that gives a machine or computer the ability to perform tasks in a way like humans. To perform ML we need to first consider a framework for development and testing. We will implement the ML code

using Python which is common for most ML applications. We also will use a modern application programming interface (API) called PyTorch [19] as the deep learning framework. This choice was made after struggling with Tensor and Keras when trying to customize the ML code. The selection of PyTorch will also support the programming of any mathematical concepts that we develop. The training and execution will be done on graphics processing units (GPUs) that support parallel processing for faster code execution that we plan to benchmark for performance.

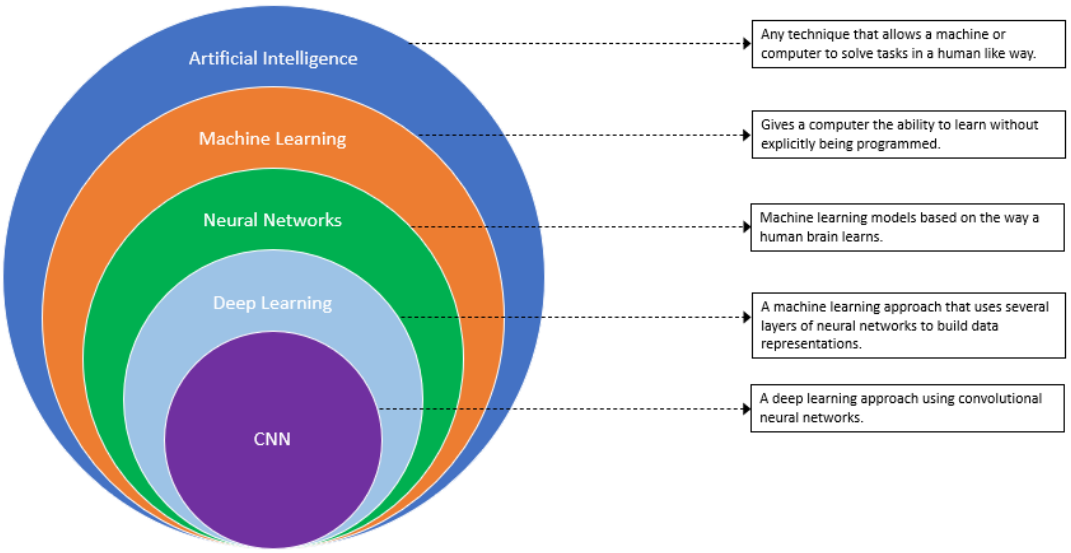


Figure 3.1: Artificial Intelligence

Recall that we are using machine learning techniques because electronic countermeasures (ECM) in general that rely on parametric modeling can fail because it violates the strict assumptions of classical signal processing algorithms. We propose to use supervised ML to label our datasets or images to then train our algorithms that will learn and predict the correct jammer classification. The reasons that follow

support the case for supervised ML:

- The problem of jammer classification is not a simple deterministic rules-based solution solved by utilizing radar parameters. You can use supervised ML to effectively solve this problem.

- Radar range-Doppler images can manifest in many ways and thus are considered a large-scale problem. Supervised ML has been proven to be an effective approach for this type of problem. [20]

We realize that there is no ideal solution and use the George Box quote "All models are wrong, but some are useful". So we propose creatively stacking simple models to realize a deep learning architecture such as CNN. Fundamentally we are seeking reduced dimensionality of our data to extract the image features which in our case are the target detections. The image features that are prominent would be the range, Doppler, and target signal strength with noise. We strive to develop a CNN that will recognize the patterns for the various DRFM jammer types described and illustrated in section 2.4.1., thus making the correct DRFM jammer type classification.

3.2 Hyperparameters

The selection and initial values of the parameters used in any machine learning problem are critical to understanding to make the best use and choice for the problem

space. We research different approaches and make recommendations that propose modifications to fit our problem space. We also propose to explore particular areas of interest to not limit or constrain ourselves to any one aspect of machine learning. The chosen areas of focus are detailed in the subsections that follow. The typical processing sequence used in machine learning for CNN motivates our areas of interest and exploration. The notional diagram below depicts the CNN processing chain which we will expand upon in the sections that follow.

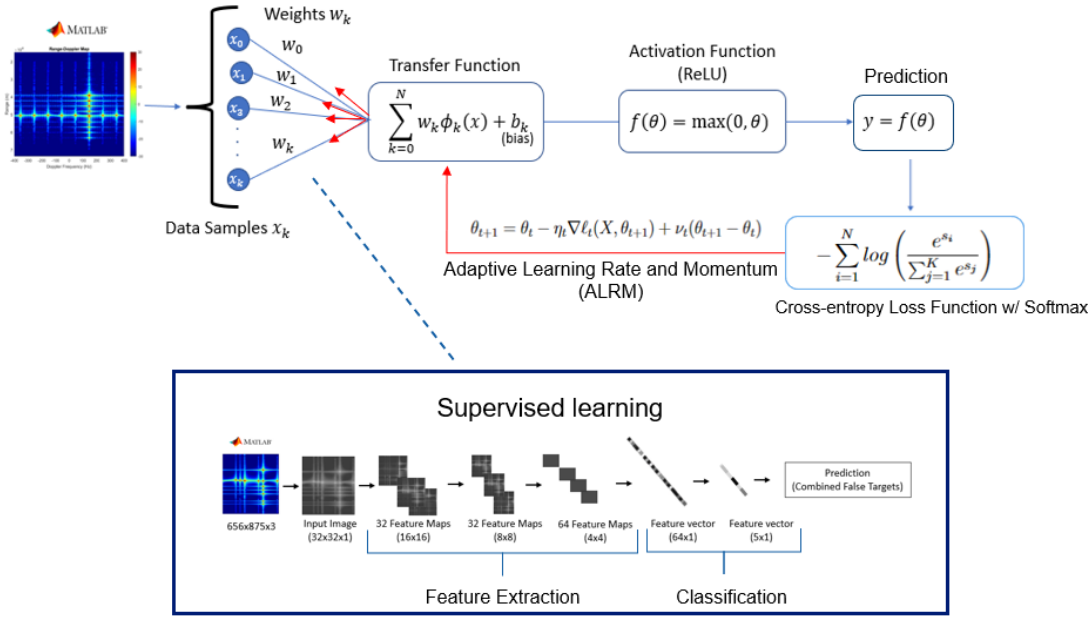


Figure 3.2: High-Level CNN Processing Chain

3.2.1 Hyperparameters Tuning

Hyperparameter tuning and machine learning in general are often done manually in a trial-and-error approach which relies on the empirical results to guide the experimenter in the optimization of the hyperparameters. This can be very time-consuming to explore the hyperparameter space. There are two popular choices for a model and a brute force approach like Grid Search could be used by defining a search space over all the hyperparameters and going through every position in the grid which is not efficient. Random Search is the other approach where random values are chosen so there is the possibility of discovery but you could also miss important settings. So we must carefully define and constrain our model. We explore several key hyperparameters to optimize our tuning and results. A mathematical approach will be developed in this chapter to gain a more rigorous understanding for better-informed choices when conducting our experiments.

Hyperparameter	Description	Value
Number of Filters	Used to map the activations from one layer to the next. The filter contains the trained weights and matches the kernel size.	[16,32,64,96]
Kernel Size	Generalized sliding dot product or convolution operation into higher dimensional spaces that are linearly separable. We convolve the filter(nxn kernel size) with the image.	[3,4,5]
Batch Size	Mini-batch size from our image dataset	[32-256]
Activation Function	Transforms our dot product summations into the final output feature vector.	ReLU with softmax
Number of Layers	The number of convolutional layers	[3-5]
Optimizer	Customized SGD (see section 3.3)	ALRM
Learning Rate	Quasi-Newton method (see section 3.3.1)	Trained automatically through ALRM
Momentum	Stochastic heavy ball method (see section 3.3.2)	Trained automatically through ALRM
Max Pooling	Creates a down-sampled feature map based on the largest segment of each feature map.	True or False
Dropout	Randomly discards nodes by setting them to zero. This helps to regularize which helps to prevent overfitting.	True or False

Figure 3.3: Hyperparameters

3.2.2 Activation Function

The activation function is responsible for the output of a node, in our case a node is each pixel for our Range-Doppler image. Each image is a 32x32 square matrix so that is 1024 input nodes in total. The question answered here "Is this pixel useful or not?", if useful then that pixel gets activated "fired like in the brain" and considered important, in a sense it's like a mathematical filter or gate. The input to the activation function is the image matrix that has gone through convolution. In this case, we are seeking a non-linear activation function with the purpose to add non-linearity to the NN, without the activation function it would be linear by design because all the hidden layers will behave in the same way because the addition of two or more linear functions is linear. In modern neural networks, the default recommendation is to

use the rectified linear unit or ReLu. [21] That however does not preclude us from gaining a mathematical perspective and selecting the proper activation function for this problem.

A nonlinear activation function $f()$ sometimes called a transfer function [22] can be expressed as

$$y(x, w) = f\left(\sum_{k=1}^{N} w_k \phi_k(x)\right) = \mathbf{w}^T \phi(x) \tag{3.1}$$

Where N is the total number of parameters in the model. We now extend this function by making the basis functions $\phi_k(x)$ or features depend on N parameters, along with adjusting the weighing coefficients w_k during the training of our CNN model. This then gives us a general neural network model that can be expressed as

$$a_k = \sum_{l=1}^{M} w_{kl} x_l + b_l \tag{3.2}$$

where b_0 is the bias constant and a_k is our activations. We now can use a transformation assuming a differentiable non-linear activation function $h_j = g(a_k)$ to then give the output of each activation.

$$a_i^{(1)} = \sum_{j=1}^{M} w_{ij}^{(1)} h_j + b_j^{(1)} \tag{3.3}$$

We use the notation $a_i^{(1,2,\ldots)}$ to denote the layers of our network. There can be multiple

layers which are common for CNNs. Written in matrix and vector form we get

$$\sum_{j=1}^{M} w_{ij}^{(1)} h_j + b_j = \begin{bmatrix} w_{0,0}^{(1)} & w_{0,1}^{(1)} & ... & w_{0,j}^{(1)} \\ w_{1,0}^{(1)} & w_{1,1}^{(1)} & ... & w_{1,j}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0}^{(1)} & w_{i,1}^{(1)} & ... & w_{i,j}^{(1)} \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ \vdots \\ h_j \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_j \end{bmatrix} \tag{3.4}$$

Written in a compact form we get

$$\mathbf{a^{(1)}} = f(\mathbf{W}^{(1)}\mathbf{h} + \mathbf{b}) \tag{3.5}$$

This now makes it convenient for writing a code block as this simply can be written for example in Python as $a = relu(dot(w, h) + b)$ where $relu$ is the rectified linear unit activation function.

Now that we have some mathematical insight into the activation function, we can proceed with an approach for selecting the best activation function for the problem at hand which we consider as a categorical image classification. Since our problem involves multiple classifications we can eliminate any activation function that is binary or is susceptible to vanish gradients. So for example the binary activation functions like binary step and sign functions would not be useful. Functions such as tanh and sigmoid can saturate or cause vanishing gradients which is not what we want when

filtering through data. We now define requirements for a good activation function for our image classification.

- Nonlinear activation function since we have a multi-layer network
- Differentiable and monotonic to allow backpropagation
- Speed and accuracy
- Solve vanishing gradient or a saturation problem

After conducting research ReLU seems to be the best choice, although we need to consider the negative values mapping to zero (i.e. dead neurons). ReLU is based on a cortex-inspired silicon circuit. [23] It has been demonstrated to improve the training of deep neural networks for deep learning. [24] We define the ReLU function as follows:

$$f(\theta) = max(0, \theta) = \begin{cases} \theta & : \ \theta \geq 0 \\ -\theta & : \ \theta < 0 \end{cases} \tag{3.6}$$

$$f'(\theta) = \begin{cases} 1 & : \ \theta > 0 \\ 0 & : \ \theta < 0 \end{cases} \tag{3.7}$$

We note that ReLU is not differentiable at zero. We can arbitrarily assume when $\theta = 0$ that the differentiated value is zero. Also, negative values result in a dead neuron since $f'(\theta) = 0$ for $\theta < 0$ such that no gradients flow back through the CNN processing chain. The consequence of pushing too many negative values through the

processing chain is that our model will stop learning since the activation is zero. To address this issue we can facilitate a slight curve or linear slope thus the negative values result in $f'(\theta) < 0$. Another approach is to lower the learning rate which we will explore in section 3.2.4.1.

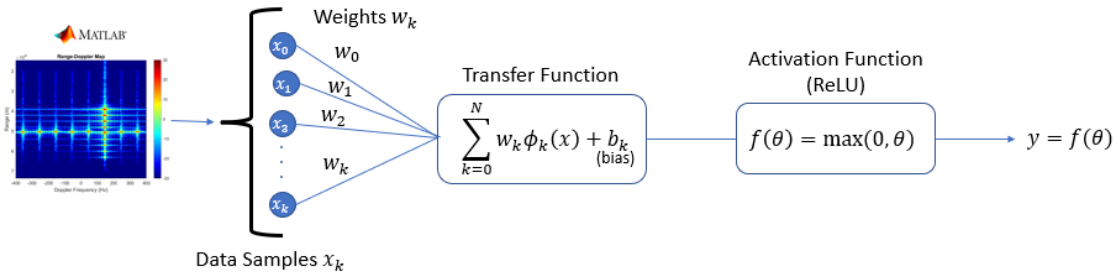


Figure 3.4: ReLU Data Flow

3.2.3 Loss Function

The function that we minimize or maximize is typically called the objective function. Since we are minimizing over some objective function and extracting an error measurement, we will refer to this as the loss function. The loss function is used to evaluate the model predictions versus the truth or input data, so this is performed at the output layer or end of the neural network. The loss function acts as a guide to the optimizer which attempts to provide the correct direction towards the global minimum. The output is a scalar number that is used to minimize or maximize the model loss, which allows us to rank or compare candidate solutions. The question

answered here is "How good or useful is my model at making predictions using a given set of parameters (i.e. weights and biases)?" Is it "Too good to be true!" which is overfitting and not useful at all. Which loss function do we use? There are many different loss functions that can be used to provide error estimates for each set of parameter weights in a CNN. Cross-entropy is the default loss function to use for most multi-class classification problems. Cross-entropy measures the error between two probability distributions. We consider the predicted model and the true training probability distributions. We seek to minimize the cross-entropy between these two distributions. A nice feature of a cross-entropy approach is that it heavily penalizes wrong predictions that have high confidence. We also propose a constraint of a symmetric loss. The claim is that we do not always expect clean data and thus must account for noisy data in our classification. It is also highly desirable to learn from corrupted data since we expect this in real-world applications.

We define our loss model as

$$\mathcal{L}(X, \theta) = -\sum_{j=1}^{N} X_j log(\theta_j) \tag{3.8}$$

where X is a random variable whose domain is our dataset of range-Doppler samples and θ between $(0, 1]$ is the predicted variable of interest. We apply a negative sign since the $log(\theta_j)$ is a negative value thus giving the positive probability that we desire. Since we have multiple classes we need a function to keep track of the probabilities

of each class. Also the output of $\mathcal{L}(X, \theta)$ is a vector of real numbers, we need a way to convert that to probabilities. The common approach is to use a softmax function which outputs a vector of sample values that sum up to a value of 1, which is generating the predicted probability distribution of the classes we define. Ideally, the aim is to have the samples to be as close to unity as possible so then we have the highest probability of the correct classification. We define our softmax function as $p : \mathbb{R}^N \to [0, 1]^N$ where $N \geq 1$ such that

$$p_i = \frac{e^{s_i}}{\sum_{j=1}^{N} e^{s_j}} \tag{3.9}$$

where $i = 1, 2, ...N$ and $\mathbf{s} = [s_1, s_2, ...s_N] \in \mathbb{R}^N$

Using our logarithmic measure from Eq.(3.8) we now have our general equation for the loss function.

$$-\sum_{i=1}^{N} log\left(\frac{e^{s_i}}{\sum_{j=1}^{K} e^{s_j}}\right) \tag{3.10}$$

3.2.4 Optimization

We define our problem as non-convex which fits for a stochastic gradient descent (SGD) based optimization approach. This implies that we seek an optimization that is not based on a direct calculation of the gradient. The SGD approach aligns with real-world types of problems where analytical solutions are very few. We also confine

our focus to mini-batch SGD which is the popular choice for training a CNN. Online training was considered but seems impractical to adjust the gradient after each forward and backward pass of the data thus affecting overall performance. Using the mini-batch approach allows each subset of data to be trained with the gradient estimated at each step. The motivation is twofold. First, we are not loading the data all at one time thus reducing the memory footprint. In our problem space that amounts to at most $(32x32x50,000)$ data samples which for the CIFAR-10 training dataset. Second, we are updating the parameters more frequently with mini-batch by running numerous batches compared to a single batch. Therefore we reduce the probability of getting stuck in a local minima and increase the probability of finding a global minima. Compare this to a full batch that tends to get stuck in local minima thus not optimal for global solutions. [25] An additional consideration is the speeding up of the convergence. Learning rate and momentum methods can be used to accelerate the solution convergence compared to default SGD. For optimization, we will establish a customized SGD approach that fits our problem space of range-Doppler imaging classification. The overall goal of our optimization process is to fine-tune the model weights to include bias to therefore have an optimized model that makes the most correct predictions for the given image test data set.

3.2.4.1 Learning Rate and Momentum

Learning rates matter and its regarded among practitioners to be the most important hyperparameter to tune. [26] [21] Too large a step and we diverge, too small of a step and we do not make progress. So if I'm confident about the gradient then for example we can set the learning rate = 1.0 and take a confident large step forward. If it's uncertain then a smaller step or learning rate is required but now it takes longer to converge to a global minimum or worse get stuck in less desirable local minima. How do we make the optimal choice for the learning rate? There are two popular choices to consider they are learning rate schedulers and adaptive learning rate methods. We seek an adaptive method. Using an adaptive approach we hope to optimize the step size to avoid small local minima and thus converge faster and have more accurate learning models that seek a global minima. We propose to combine an adaptive learning rate and momentum so as not to concern the user with having to experiment with setting these hyperparameters to obtain the best results. Recent activity in this area has motivated research in finding optimal solutions for this set of hyperparameters [27] [28] and thus is the foundation for a new algorithm that we name the Adaptive Learning Rate and Momentum (ALRM) method. Our high-level goals for this method are to reduce the computational complexity, faster convergence to a global minimum with auto-tuning parameters that provide numerical stability, and easy implementation of the algorithms.

3.3 ALRM Method

We start with a default expression for SGD where θ_t is our model variable parameter of interest that are more precisely mathematical tensors which are estimated by our equation, X_t is a random variable and η_t is the adaptive learning rate we evaluate at each iteration using the training image data as input:

$$\theta_{t+1} = \theta_t - \eta_t \nabla \mathcal{L}(X_t, \theta_t) \tag{3.11}$$

Research points out a sensitivity to the learning rate for default SGD. [29] [30] So we propose an implicit SGD approach that reduces learning rate sensitivity and provides numerical stability. Implicit updates are simply evaluated at the next iteration $(t+1)$ rather than the current iteration (t). We define this as implicit because we have θ_{t+1} on both sides of the equation. So then we have

$$\theta_{t+1} = \theta_t - \eta_t \nabla \mathcal{L}(X_t, \theta_{t+1}) \tag{3.12}$$

Expanding on this implicit Eq.(3.12) to gain more intuition and motivate numerical stability [31], let's suppose a limit that approaches infinity for our model vector parameter θ such that

$$\lim_{n \to \infty} \theta_{t+1}^{(n)} \quad \text{where } n \in \mathbb{Z}^{0+} \tag{3.13}$$

Now let's write out the sequence

$$\theta_{t+1}^{(1)} = \theta_t - \eta_t \nabla \mathcal{L}(X_t, \theta_t^{(0)})$$

$$\theta_{t+1}^{(2)} = \theta_t - \eta_t \nabla \mathcal{L}(X_t, \theta_{t+1}^{(1)})$$

$$\ldots$$

$$\theta_{t+1}^{(\infty)} = \theta_t - \eta_t \nabla \mathcal{L}(X_t, \theta_{t+1}^{(\infty)}).$$

We can observe that the last term of the sequence above can be re-written as $\theta_{t+1} = \theta_t - \eta_t \nabla \mathcal{L}(X_t, \theta_{t+1})$ where we assume that the limit converges to a fixed point. We then have the implicit SGD we defined in Eq. (3.12). Therefore the implicit SGD is shown to be a repeated sequence or variant of the default expression for SGD. It's easy to see that we are just updating the θ_t term until a fixed point or optimal solution is reached. This concept is related to the *self-consistency principle* in statistics. The term "self-consistency" was introduced in 1989 by Hastie and Stuetzle, where given a smooth curve or surface each point is considered the mean of all points that project orthogonally onto it. We now provide a mathematical argument for

numerical stability by re-writing our implicit Eq.(3.12).

$$\begin{aligned}
\theta_{t+1} &= \theta_t - \eta_t \nabla \mathcal{L}(X_t, \theta_t), \\
\theta_t &= \theta_{t+1} + \eta_t \nabla \mathcal{L}(X_t, \theta_t), \\
||\theta_t - \theta^*||^2 &\geq ||\theta_{t+1} - \theta^*||^2 + 2\eta_t(\theta_{t+1} - \theta^*)^T \nabla \mathcal{L}(X_t, \theta_t), \\
||\theta_t - \theta^*||^2 &\geq (1 + \eta_t \mu)||\theta_{t+1} - \theta^*||^2 \quad \text{where } \mu > 0, \\
||\theta_{t+1} - \theta^*||^2 &\leq \frac{1}{(1 + \eta_t \mu)}||\theta_t - \theta^*||^2.
\end{aligned}$$

This shows that $||\theta_{t+1} - \theta^*||^2$ has a high probability of contracting and thus providing the numerical stability that we desire.

A critical operation is finding the weighted averages of our function. So we define a finite-sum structure by minimizing over our loss function $\mathcal{L}(X, \theta)$ that is continuously differentiable over a dataset space called S where $\mathcal{L}(X, \theta)$ is defined by

$$\mathcal{L}(X, \theta) : [S \subset \mathbb{R}^n \to \mathbb{R}] \quad \text{where } n \text{ is the number of sample data points} \tag{3.14}$$

We consider an unconstrained optimization in which we seek the minimization of

$$\theta^* = \min_{\theta \in \mathbb{R}^n} \mathcal{L}(X, \theta) = \frac{1}{n} \sum_{t=1}^{n} \ell_t(X, \theta) \tag{3.15}$$

$$\approx \mathbb{E}[\ell(X, \theta)] \tag{3.16}$$

such that a sample data point θ^* is a local minimum if $\mathcal{L}(\theta^*) \leq \mathcal{L}(X, \theta) \forall \in S$. Furthermore we make the following assumptions for non-convex stochastic optimization which are typical for this type of problem.

- Assumption 1: The stochastic gradient is uniformly bounded, for example, $sup_t(||g^t||) \leq R$ with $R > 0$.
- Assumption 2: The SGD is an unbiased estimate, for example, $\mathbb{E}[\ell(X, \theta)] = \nabla \mathcal{L}(X, \theta_{t+1})$.
- Assumption 3: The gradient of $\mathcal{L}$ is $L - Lipschitz$, for example, $||\nabla \mathcal{L}(x) - \nabla \mathcal{L}(y) \leq L||x - y||$ where $L > 0$.

3.3.1 Learning Rate Dynamic Adjustment

We propose using a Quasi-Newton method which is a well-established approach to solving a non-convex optimization problem. This is based on Newton's method which is an alternative that assumes that the Hessian matrix is not available or practical since its computational complexity is $\mathcal{O}(n^3)$ to compute the Hessian matrix and its inverse. So then the idea is to then approximate the Hessian (and its inverse) matrix. We will start with the BFGS method which is named after the authors Broyden, Fletcher, Goldfarb, and Shanno and is considered a very effective Quasi-Newton method and its computational complexity $\mathcal{O}(n^2)$ is less expensive to compute

and given by [32]:

$$B_{k+1} = B_k - \frac{1}{\mathbf{s}_k^T B_k \mathbf{s}_k} B_k \mathbf{s}_k \mathbf{s}_k^T B_k + \frac{1}{\mathbf{y}_k^T \mathbf{s_k}} \mathbf{y_k} \mathbf{y}_k^T \tag{3.17}$$

where

$$y_k = \nabla \mathcal{L}(\theta_{k+1}) - \nabla \mathcal{L}(\theta_k) \text{ and } s_k = \theta_{k+1} - \theta_k.$$

The BFGS method provides positive-definite solutions so then we define B_k as the Hessian estimate which is a positive definite symmetric nxn matrix, furthermore the matrix B_{k+1} is guaranteed to be positive definite given B_k is also positive definite and we satisfy the curvature condition of $\mathbf{s}_k^T \mathbf{y}_k > 0$. There is still a performance concern if the rank of the matrix B_k is high. The optimization problem we are solving assumes to be unconstrained since we are required to solve for $\nabla \mathcal{L}(\theta_k)$ at each iteration. There are methods that we recommend to construct a sequence of lower-rank matrices that have been proven to do a good job of approximating the Hessian matrix and thus provide the improved computational performance we seek. The limited-memory BFGS (L-BFGS) is that method [33]. So then we continue with Eq. (3.16) and let us define S_k and Y_k by $(n \times k)$ matrices where

$$S_k \triangleq [\mathbf{s}_0, \mathbf{s}_1, ..., \mathbf{s}_{k-1}] \tag{3.18}$$

$$Y_k \triangleq [\mathbf{y}_0, \mathbf{y}_1, ..., \mathbf{y}_{k-1}] \tag{3.19}$$

Then let B_0 be a positive definite symmetric nxn matrix such that $s_i^T y_i > 0$ then using Eq. (3.16) we get a more compact form

$$B_k = B_0 - [B_0 S_k \; Y_k] \begin{bmatrix} S_k^T B_0 S_k & L_k \\ L_k^T & -D_k \end{bmatrix}^{-1} \begin{bmatrix} S_k^T B_0 \\ Y_k^T \end{bmatrix} \tag{3.20}$$

where D_k is the diagonal part and L_k the lower triangular part of the matrix we define as a symmetric $k \times k$ matrix

$$(L_k)_{i,j} = \begin{cases} s_{i-1}^T y_{j-1} & if \; i > j \\ 0 & otherwise. \end{cases} \tag{3.21}$$

We now take the compact form of the BFGS Eq. (3.19) and define it as L-BFGS. Let $B_0 = \sigma I$ where I is an identity matrix and σ is a positive scale value. Substituting into Eq. (3.20) we then get

$$B_k = \sigma I - [\sigma S_k \; Y_k] \begin{bmatrix} S_k^T \sigma S_k & L_k \\ L_k^T & -D_k \end{bmatrix}^{-1} \begin{bmatrix} S_k^T \sigma \\ Y_k^T \end{bmatrix} \tag{3.22}$$

We now provide the inverse of the Hessian estimate H_k where $H_k = B_k^{-1}$, so then we simply get

$$H_{k+1} = \left(I - \frac{y_k s_k^T}{y_k^T s_k} \right) H_k \left(I - \frac{s_k y_k^T}{y_k s_k^T} \right) + \frac{y_k y_k^T}{y_k s_k^T} \tag{3.23}$$

It's important to point out the main weakness we discovered in the L-BFGS method. If we have a Hessian matrix that contains a wide variance of eigenvalues, the solution tends to converge more slowly due to the ill-conditioned problem. This approach is still favorable over a basic gradient descent since it has the property of always monotonically decreasing at each iteration unless the model parameter θ has arrived at a local or global minimum. Also keeping in mind one of our goals for ALRM is for easy implementation. Code libraries exist for the L-BFGS method through PyTorch, which aligns with our coding implementation strategy. We do however need to modify and extend the code to implement our ALRM method.

3.3.2 Momentum Dynamic Adjustment

The use of stochastic momentum with SGD is a popular choice and widely accepted which we will explore. In particular, we consider a stochastic heavy ball momentum with Polyak averaging or Polyak's momentum. [34] The motivation is to escape the local minima or saddle points which can inhibit the path to a global minimum. We did consider Nesterov momentum which is also a popular choice and is an option in the popular Adam optimizer. Since we are using a finite sum structure recent papers have proved a possible divergence with Nesterov's approach. [35] [36]

First, we assert that a weighted average of stochastic gradients will be preserved at each iteration through our sample data, commonly referred to as Polyak-Ruppert averaging. We also will concentrate on locating a second-order point assuming a

smooth non-convex optimization by implicit SGD using a *stochastic heavy ball method* which we first established by the *heavy ball method* which is defined by the following state transitions:

$$p_t = \nabla \mathcal{L}(X, \theta_{t+1}) \mid \nu_t p_{t+1} \tag{3.24}$$

$$\theta_{t+1} = \theta_t + \eta_t p_t \tag{3.25}$$

We can rewrite this as the iteration using our general Eq.(3.12) that we derived earlier.

$$\theta_{t+1} = \theta_t - \eta_t \nabla \ell_t(X, \theta_{t+1}) + \nu_t(\theta_{t+1} - \theta_t) \tag{3.26}$$

where the term $\nu_t(\theta_{t+1} - \theta_t)$ is referred to as *momentum*. Note that we replaced the true gradient $\nabla \mathcal{L}(\theta_{t+1})$ with our unbiased estimate gradient from *assumption 2* where $\nabla \ell_t = \mathbb{E}[\ell(X, \theta)]$ to simply make this a *stochastic heavy ball method*. We now have a general Eq.(3.26) for the learning rate η_t combined with momentum ν_t.

3.3.3 Algorithms

The algorithms in this section are implemented in Python using PyTorch. [19]

Algorithm 1 *ALRM*, our new algorithm for SGD that incorporates adaptive methods for learning rate and momentum. Vector operation is element-by-element.

Require: $\nabla \ell_t(X, \theta_{t+1})$: SGD loss function
Require: X : Random variable
Require: θ : Model parameter weights ▷ Variable column vector of interest
Require: ϵ : Convergence tolerance
Require: η : Learning rate
Require: ν : Momentum
Require: m : Mini-batch size
Ensure: $\epsilon > 0$

Input: A set of training data that consists of training vectors $\{x_n\}$ where $n = \{1, 2, ..., N\}$.
function ALRM(a, b)
  for $t = 0, ..., T$ **do**
    Choose m integers $k_1, k_2, ..., k_m$ uniformly and independent from {1,2,3,...,N}
    $s_t \leftarrow$ Sample a mini-batch of a set of training data
    while $n_i < n_{max}$ **do** ▷ Quasi-Newton optimization step condition
      $\eta \leftarrow \beta \cdot \eta$ ▷ η will be smaller by at most a factor of β
      perform Quasi-Newton method ▷ See Algorithm 2
      if $abs(\nabla \ell_t(X, \theta_{t+1}) - \nabla \ell_t(X, \theta_t)) < \eta$ **then**
        perform stochastic heavy-ball method ▷ See Algorithm 3
        $\theta_{t+1} \leftarrow \theta_t - \eta_t \nabla \ell_t(X, \theta_{t+1}) + \nu_t(\theta_{t+1} - \theta_t)$
      end if
      if $abs(\sum \nabla \ell_t(X, \theta_{t+1})) \leq \epsilon$ **then**
        return break
      end if
      if $\nabla \ell_t(X, \theta_{t+1}) \leq \epsilon$ **then**
        return break
      end if
      if $\nabla \ell_t(X, \theta_{t+1}) - \nabla \ell_t(X, \theta_t) \leq \epsilon$ **then**
        return break
      end if
    end while
    $\theta_{\mathbf{t+1}} \leftarrow \hat{\theta}_{\mathbf{t}}$
  end for
  return $\theta_{\mathbf{t+1}}$
end function

Algorithm 2 *Quasi-Newton Method based on L-BFGS*, solve a non-linear optimization problem with an approximation of the Hessian matrix and its inverse.

Require: $\nabla\ell(\theta)$: Gradient loss function
function L-BFGS(a, b)
    Choose an initial Hessian inverse H_t^0 ▷ cite
    $p_k \leftarrow -H_t \nabla\ell(X, \theta_{t+1})$ ▷ compute a search direction p_t
    $x_{t+1} \leftarrow x_t + \eta_t p_t$ ▷ η satisfies the Wolfe conditions(cite)
    if t>m **then**
        $\{s_{t-m}, y_{t-m}\}$ ▷ delete from memory
    end if
    $s_t \leftarrow x_{t+1} - x_t$ ▷ save in memory
    $y_t \leftarrow \nabla\ell(\theta_{t+1}) - \nabla\ell(\theta_t)$ ▷ save in memory
    $t \leftarrow t + 1$
    if $\sim$ (converged || failed) **then**
        return inverse Hessain updated state
    end if
end function

Algorithm 3 *Momentum Method*, a stochastic heavy ball method based on Polyak momentum.

Require: $0 \leq \nu \leq 1$: Momentum
Require: $0 \leq \eta \leq 1$: Learning rate (i.e. Step size)
function SHB(a,b)
    for $t = 0, ..., T$ **do**
        $\theta_{t+1} = \theta_t - \eta_t \nabla\ell_t(X, \theta_{t+1}) + \nu_t(\theta_{t+1} - \theta_t)$
    end for
    return ν
end function

Chapter 4

Models

4.1 Overview

Models rooted in the MATLAB and Python code base are used for conducting experiments to determine the goodness of our CNN model with the proposed ALRM method.

4.2 Radar Simulation Model

The radar model is implemented in MATLAB. The selection of MATLAB is based on several reasons. First, many professional radar engineers rely on MATLAB because of its rich and vetted radar functions along with the toolboxes that model radar systems and handle radar signal processing. Secondly, I have used MATLAB for

many years and I trust the software for modeling radar systems. To ensure the radar model does what we need the following requirements are established and followed.

4.2.1 Requirements

1. The radar testbed shall simulate the radar transmit and receiver operations.
2. The radar testbed shall model environmental effects.
3. The radar testbed shall model antenna effects.
4. The radar testbed shall provide a waveform model.
5. The radar testbed shall generate DRFM false targets.
6. The radar testbed shall generate real targets.
7. The radar testbed shall generate noise interference.
8. The radar testbed shall generate raw uncompressed IQ data.
9. The radar testbed shall perform signal processing on the IQ signal.
10. The radar testbed shall output range Doppler maps.

4.2.2 Data Generation

The main objective of the radar simulation model is to produce realistic range-Doppler images used as input to our CNN model. The images are 32x32 in size, which

represents the spatial location and 256 grayscale values that range from (0:black to 255:white) for each pixel in the image. While color images with a size of 656x875x3 could have been used and visually look more appealing it introduces another dimension. The extra-dimensional increases computational complexity, along with the higher resolution. This is not needed since we are interested in the intensity of each pixel value, therefore grayscale works just fine, so we down-sampled the images to a size of 256x256 in grayscale. It is important to point out that the input layer has a depth of 1, but later layers through convolution still have a 2D spatial representation but include a depth that corresponds to the number of features (not the number of channels, like grayscale=1 and RGB=3), thus the convolutional layer outputs are 3D. Several different image sizes were visually inspected. The sizes ranged from 16x16 up to 256x256. The selection of 32x32 with 1024 numerical values should have enough resolution to by using the CNN and produce reasonable results. It also aligns with our CIFAR-10 baseline dataset in terms of the matrix size. Too much resolution just introduces computational overhead and does not improve results. We also constrain our range-Doppler images to square matrices which allows for useful linear algebra operations such as calculating the determinant, eigenvalues, eigenvectors, and singular value decomposition (SVD). Since the range-Doppler images are dominated mostly by Gaussian noise the matrices are also dense and thus mostly non-zero.

4.3 CNN Model

The central objective of our CNN model is best depicted below in the notional Figure 4.2 as to how our image classification experiments will work. The idea is to reduce the image dimensionality in layers as shown to obtain a final output feature vector that is made up of a prediction of the defined classes. Strictly speaking, this is the output of the softmax function defined in Eq. (3.9).

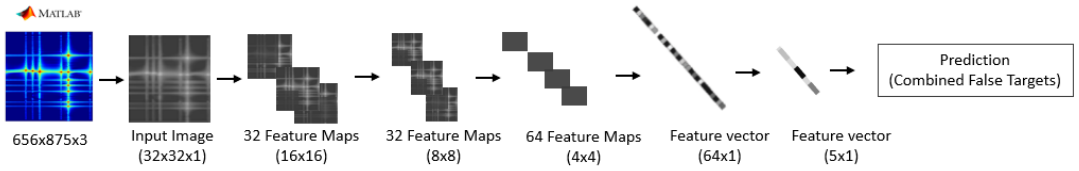


Figure 4.1: CNN Feature Map Flow

The learning style will be to map the images to the output which is a supervised learning technique. This technique allows us to provide labels and categorize the images into unique classes. We provided an overview of the DRFM classes starting in section 2.4.1.1.

4.3.1 Architecture

There are many different types of CNNs and architectures to exploit. While it's easy to understand the choice of the final output (i.e. feature vector of classes) the

structure preceding and its path is complex. We did explore hyperparameters that help us along the path and in some respects create a path through SGD. There is still the problem and decision of how many layers, the number of nodes in each layer, and how to connect it all. They are guidelines and research that suggests various architectures but there appears to be no widely accepted architecture for image classification. In other words, it's not an exact science. We put this in the bucket of the "no free lunch" (NFL) theorem. [37] Essentially the NFL theorem states that there is no single learning solution that is "one size fits all" on the path to the best architecture for image classification. Just because something works well on one type of problem does not guarantee an ideal transfer to problems of a different type, which is often the case. So the best we can hope for is that our CNN model works in a relatively restricted set of problems. So faced with which CNN architecture is the best, what to do? We will use a conventional approach and experiment with different architectures and choose the one that is the best for the validation and test data set. This is still at best an empirical approach to the best solution.

We first make a general assumption for the CNN architecture. Since the input range-Doppler images are square (32x32) we assume the CNN architecture to be symmetric, so both dimensions have the same value for all variables. Otherwise, if the architecture or the images are asymmetric then we would have to calculate the attributes for the feature map at each dimension. We begin by breaking the CNN architecture into the following hyperparameters table and then propose an empirical

method to find the best set of hyperparameter values. As mentioned in section 3.2.1 Grid Search and Random Search can be used but have their drawbacks. So we propose to make informed constraints on the hyperparameters, such that it limits our choices in the best way. We realize that this approach is not optimal so we will try and proceed with caution.

Hyperparameter	Description	Value
Number of Filters	Used to map the activations from one layer to the next. The filter contains the trained weights and matches the kernel size.	[16,32,64,96]
Kernel Size	Generalized sliding dot product or convolution operation into higher dimensional spaces that are linearly separable. We convolve the filter(nxn kernel size) with the image.	[3,4,5]
Batch Size	Mini-batch size from our image dataset	[32-256]
Activation Function	Transforms our dot product summations into the final output feature vector.	ReLU with softmax
Number of Layers	The number of convolutional layers	[3-5]
Optimizer	Customized SGD (see section 3.3)	ALRM
Learning Rate	Quasi-Newton method (see section 3.3.1)	Trained automatically through ALRM
Momentum	Stochastic heavy ball method (see section 3.3.2)	Trained automatically through ALRM
Max Pooling	Creates a down-sampled feature map based on the largest segment of each feature map.	True or False
Dropout	Randomly discards nodes by setting them to zero. This helps to regularize which helps to prevent overfitting.	True or False

Table 4.1: Hyperparameters

We propose to constrain our choices in the best way thus limiting the different permutations that we need to test. While this seems counter-intuitive to impose these restrictions we will leverage an architecture that has substantial citations and documented success for this type of image classification problem. In 2015 Microsoft Research presented a residual learning framework to ease the training of networks

that were extensively deeper than previously used. [38] The ResNet NN allows us to train more layers by skipping over layers. It also has been tested extensively on known datasets such as CIFAR-10. So then there are established ResNet architectures for selection. These come with defined kernel sizes, many convolutional layers, max, and average pooling thus significantly reducing the hyperparameters to test. ResNet architectures work by not computing through each layer in a stack which is the conventional approach. Instead a residual mapping function $\mathcal{F}(\mathbf{x})$ is used to fit the layers. In principle, there is a desired sub-mapping that we define as $\mathcal{H}(\mathbf{x})$ where $\mathcal{F}(\mathbf{x}) := \mathcal{H}(\mathbf{x}) - \mathbf{x}$. This then is mapped into $\mathcal{F}(\mathbf{x}) + \mathbf{x}$. Figure 4.2 depicts this concept.

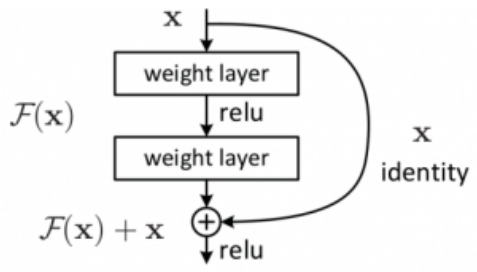


Figure 4.2: ResNet Residual Block

This is often referred to as skip connections that perform an identity mapping function that allows us to skip over the layers and use deeper neural networks without the computational cost of stacked layers. Below is a table of ResNet architectures that we will test over and then select the best candidate based on our metrics.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Table 4.2: ResNet Architectures

Chapter 5

Experiments and Results

5.1 Methodology

To initiate the experiment testing we start with a couple of non-convex functions, these are quick tests to determine that our ALRM method is viable. We will use the following non-convex functions using the ALRM method against a default SGD:

$$Rosenbrock \rightarrow f_1(x, y) = (1 - x)^2 + 100(y - x^2)^2 \tag{5.1}$$

$$Himmelblau \rightarrow f_2(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2 \tag{5.2}$$

We then follow with the established CIFAR-10 image dataset which is a typical choice for CNN practitioners. The idea is to baseline with a known good data source. This also addresses several machine-learning challenges that are important in running these

experiments.

1. Poor data quality leads to garbage-in garbage-out.

2. Variety of sample images for classification, so it's not too simple.

3. Lack of data, the CIFAR-10 dataset has an 80 million tiny images dataset and consists of 60,000 32x32 color images containing one of 10 object classes, with 6000 images per class.

4. Good representative data, to not bias the CNN model.

We iterate through the ResNet architectures in Table 4.2 and set a stopping point when the test model accuracy reaches at least 85% or stop if the model is training for more than 5 hours. At this point in the testing, we desire a good accurate model and do not want to spend too much time training our models since trends such as loss and accuracy are easy to measure and monitor. To help with overfitting we adapt a regularization method using L1 ($||x_1||$) and L2 ($||x_2||$) norms. Which are defined as

$$||x_1|| = \sum |x_i| \tag{5.3}$$

$$||x_2|| = \sqrt{\sum x_i^2} \tag{5.4}$$

Notionally we depict an overfit and a good fit in Figure 5.1 below

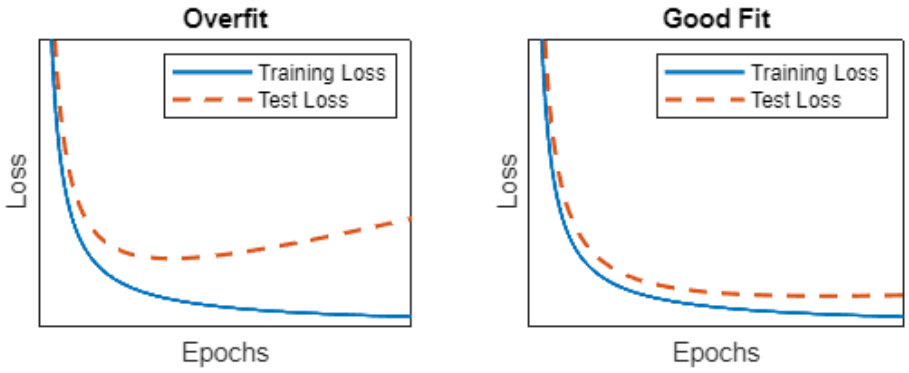


Figure 5.1: Model Fitting

For test accuracy and loss, we define the following metric scores:

Test Loss	Test Accuarcy
Cross-Entropy = 0.00: Perfect probabilities	Test Accuarcy > 95%: Excellent
Cross-Entropy < 0.02: Great probabilities	95% > Test Accuarcy > 90%: Great
Cross-Entropy < 0.05: On the right track	90% > Test Accuarcy > 85%: Very Good
Cross-Entropy < 0.20: Good	85% > Test Accuarcy > 70%: Good
Cross-Entropy > 0.30: Fair	70% > Test Accuarcy > 60%: Fair
Cross-Entropy > 1.00: Poor	50% < Test Accuarcy < 60%: Poor
Cross-Entropy > 2.00 Something is broken	Test Accuarcy < 50%: Something is broken

Table 5.1: Test Accuracy and Test Loss Scores

We then select the best ResNet candidate based on the following metrics.

Metrics	Description	Python Code
Correct Predictions	Compares the known training label to the predicted label and provides a summation of correct predictions.	(predicted==train_labels).sum() or (predicted==test_labels).sum()
Cross Entropy Loss	Cross entropy loss with softmax from CNN with applied penalty during forward propagation using a wrapped tensor and known labels.	cross_entropy(outputs,labels)+l1_penalty+l2_penalty
Outputs	Wrapped tensor <Variable()> from the forward propagation of the neural network that maintains the gradient function for backpropagation which allows the gradient calculation using the chain	net(Variable(train_images)) or net(Variable(test_images))
Overfit	The percentage between both maximum accuracy values from the training and test data.	1-torch.div(torch.max(test_acc),torch.max(training_acc),rounding_mode='trunc')
Predictions	Returns the maximum value of all elements in the tensor. In this case, for the max value in each row it provides the label prediction given its column position.	torch.max(outputs.data,1)
Test Accuracy	Percent accuaracy based on the number of correct predictions from the total inputs	torch.div(100*correct, total, rounding_mode='trunc')
Total Inputs	Scalar size of the number of images mapped to an assigned label.	train_labels.size(0) or test_labels.size(0)
Training Accuracy	Percent accuaracy based on the number of correct predictions from the total inputs	torch.div(100*correct, total, rounding_mode='trunc')
Training Time	Total training time	(time.time() - start_time)

Table 5.2: Metrics

Using the selected ResNet candidate we now have more runs varying the batch size. We propose testing batch sizes of 8, 16, 32, 64, 128, and 256, which provides a good range of batch sizes. We then follow with experiments using the range-Doppler images produced by our radar simulation model using the same approach. The selection process will be based on the metrics which are captured in the Experiment Results section 5.2.1.

5.2 Experiments

Test Case	Architecture	Dataset	Optimizer	Epochs	Batch Size
Rosenbrock_01	Initial	Function	SGD, ALRM	100	NA
Himmelblau_02	Initial	Function	SGD, ALRM	100	NA
CIFAR-10_01	ResNet18	CIFAR-10	ALRM	variable	32
CIFAR-10_02	ResNet34	CIFAR-10	ALRM	variable	32
CIFAR-10_03	ResNet50	CIFAR-10	ALRM	variable	32
CIFAR-10_04	ResNet101	CIFAR-10	ALRM	variable	32
CIFAR-10_05	ResNet152	CIFAR-10	ALRM	variable	32
CIFAR-10_06	ResNet-34	CIFAR-10	ALRM	variable	8
CIFAR-10_07	ResNet-34	CIFAR-10	ALRM	variable	16
CIFAR-10_08	ResNet-34	CIFAR-10	ALRM	variable	64
CIFAR-10_09	ResNet-34	CIFAR-10	ALRM	variable	128
CIFAR-10_10	ResNet-34	CIFAR-10	ALRM	variable	256
RD_01	ResNet-34	RD Images	ALRM	variable	8
RD_02	ResNet-34	RD Images	ALRM	variable	16
RD_03	ResNet-34	RD Images	ALRM	variable	32
RD_04	ResNet-34	RD Images	ALRM	variable	64
RD_05	ResNet-34	RD Images	ALRM	variable	128
RD_06	ResNet-34	RD Images	ALRM	variable	256

Table 5.3: Experiments

5.2.1 Experiment Results

Test Case	Architecture	Dataset	Optimizer	Epochs	Batch Size	Training Accuracy	Testing Accuracy	Training Loss	Testing Loss	Overfit	Training time (secs)	Time per Epoch(secs)
Rosenbrock_01	Initial	Function	SGD, ALRM	100	NA	NA	NA	NA	NA	NA	1.27	0.01
Himmelblau_02	Initial	Function	SGD, ALRM	100	NA	NA	NA	NA	NA	NA	1.27	0.01
CIFAR-10_01	ResNet18	CIFAR-10	ALRM	39	32	98%	84%	0.004	0.021	14%	11761	302
CIFAR-10_02	ResNet34	CIFAR-10	ALRM	13	32	96%	84%	0.009	0.027	13%	5351	412
CIFAR-10_03	ResNet50	CIFAR-10	ALRM	26	32	81%	70%	0.020	0.037	14%	10645	409
CIFAR-10_04	ResNet101	CIFAR-10	ALRM	24	32	70%	58%	0.100	0.031	17%	20500	854
CIFAR-10_05	ResNet152	CIFAR-10	ALRM	8	32	28%	28%	0.400	0.053	0%	17920	2240
CIFAR-10_06	ResNet-34	CIFAR-10	ALRM	16	8	96%	81%	0.200	0.145	16%	14973	936
CIFAR-10_07	ResNet-34	CIFAR-10	ALRM	30	16	99%	86%	0.012	0.073	13%	16281	543
CIFAR-10_08	ResNet-34	CIFAR-10	ALRM	18	64	98%	86%	0.0100	0.100	12%	5664	315
CIFAR-10_09	ResNet-34	CIFAR-10	ALRM	48	128	99%	86%	0.0008	0.009	13%	14096	294
CIFAR-10_10	ResNet-34	CIFAR-10	ALRM	14	256	98%	87%	0.0004	0.003	11%	3953	282
RD_01	ResNet-34	RD Images	ALRM	75	8	99%	80%	0.0010	0.006	19%	5826	78
RD_02	ResNet-34	RD Images	ALRM	100	16	99%	68%	0.0040	0.118	31%	2500	25
RD_03	ResNet-34	RD Images	ALRM	250	32	99%	55%	0.0020	0.173	44%	7512	30
RD_04	ResNet-34	RD Images	ALRM	100	64	99%	58%	0.0010	0.092	41%	1856	19
RD_05	ResNet-34	RD Images	ALRM	100	128	99%	59%	0.0008	0.012	40%	1743	17
RD_06	ResNet-34	RD Images	ALRM	200	256	99%	58%	0.0006	0.008	41%	3200	16

Figure 5.2: Experiment Measurements

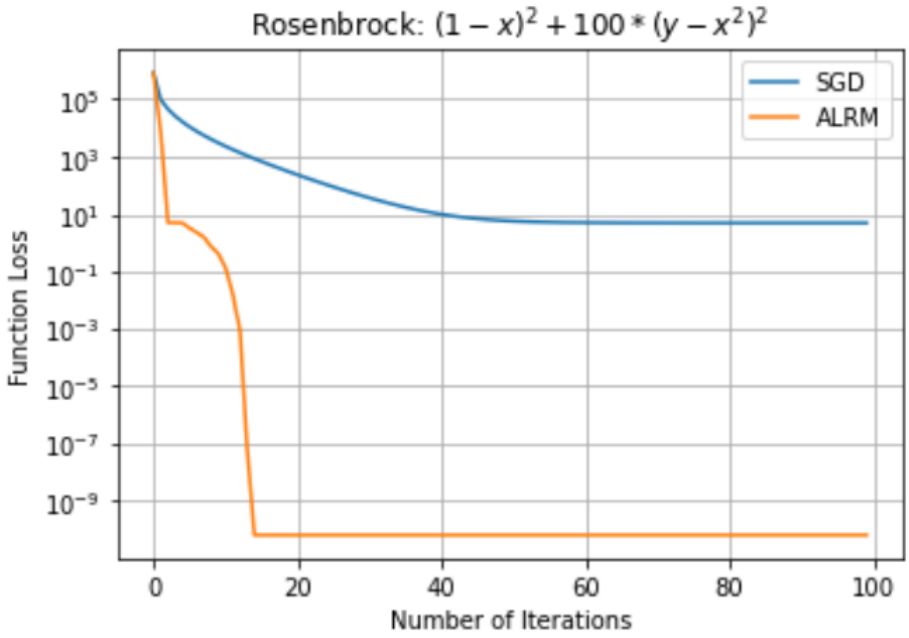


Figure 5.3: Rosenbrock Objective Loss

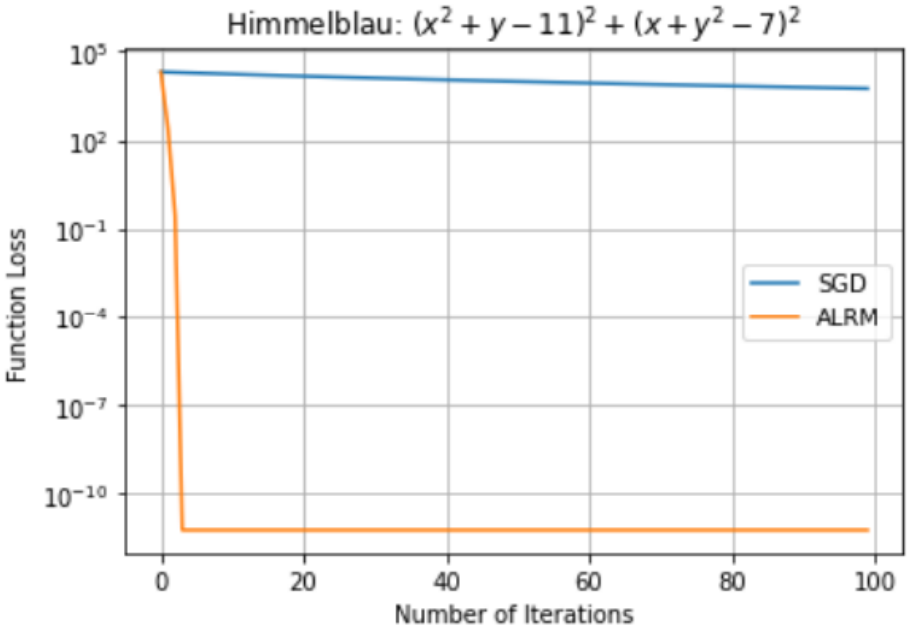


Figure 5.4: Himmelblau Objective Loss

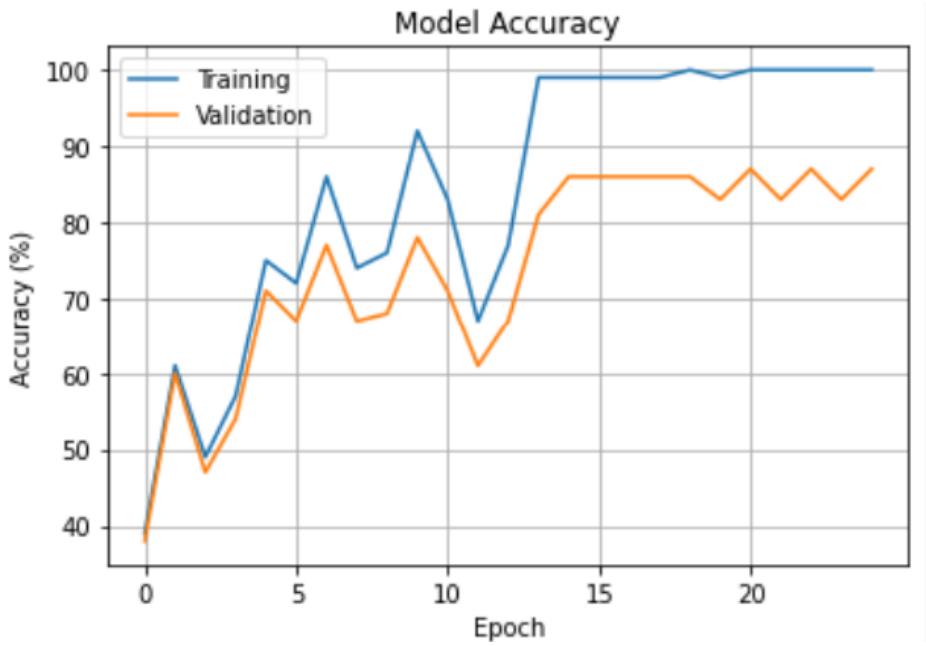


Figure 5.5: CIFAR-10 Model Accuracy

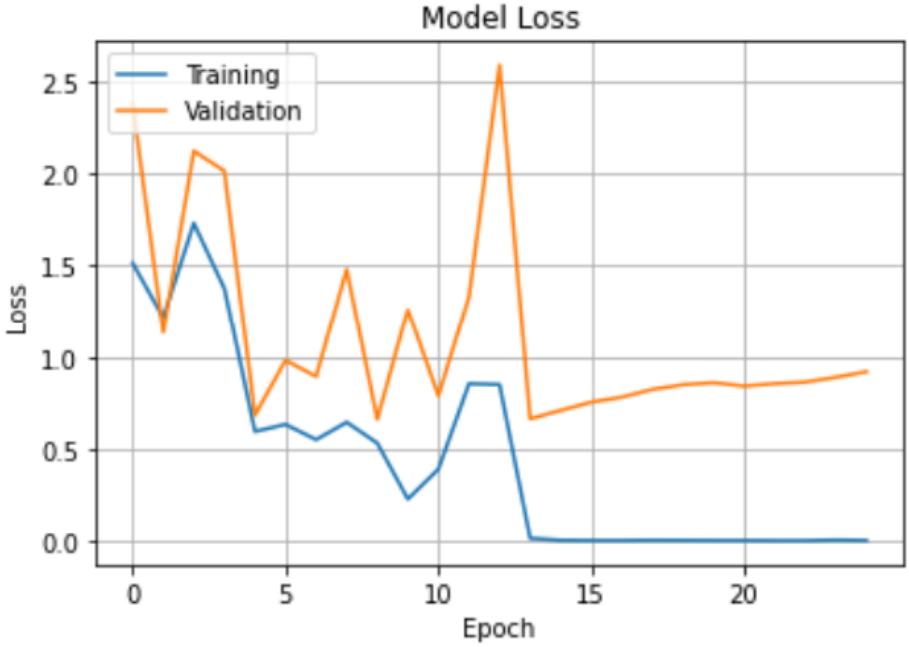


Figure 5.6: CIFAR-10 Loss

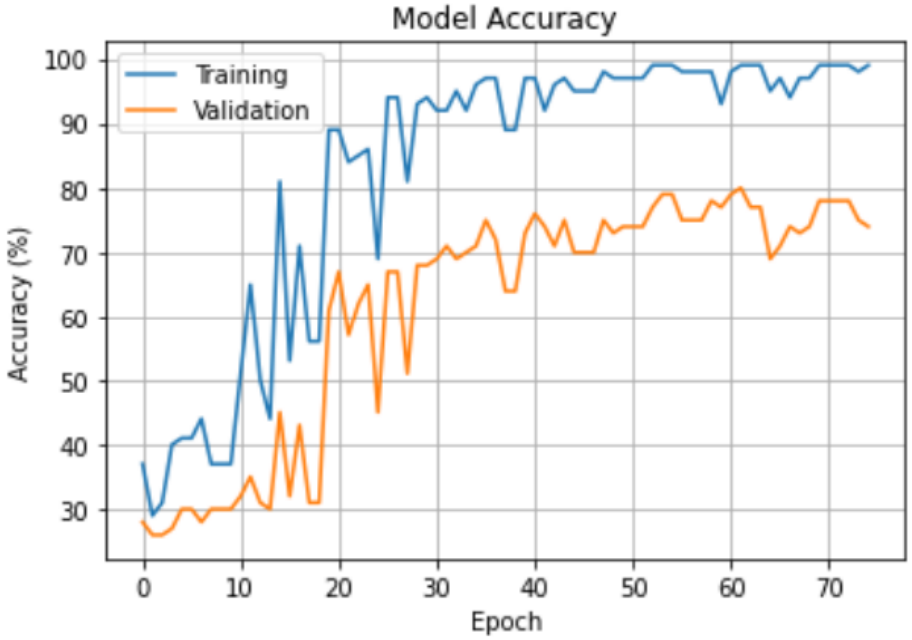


Figure 5.7: DRFM Model Accuracy

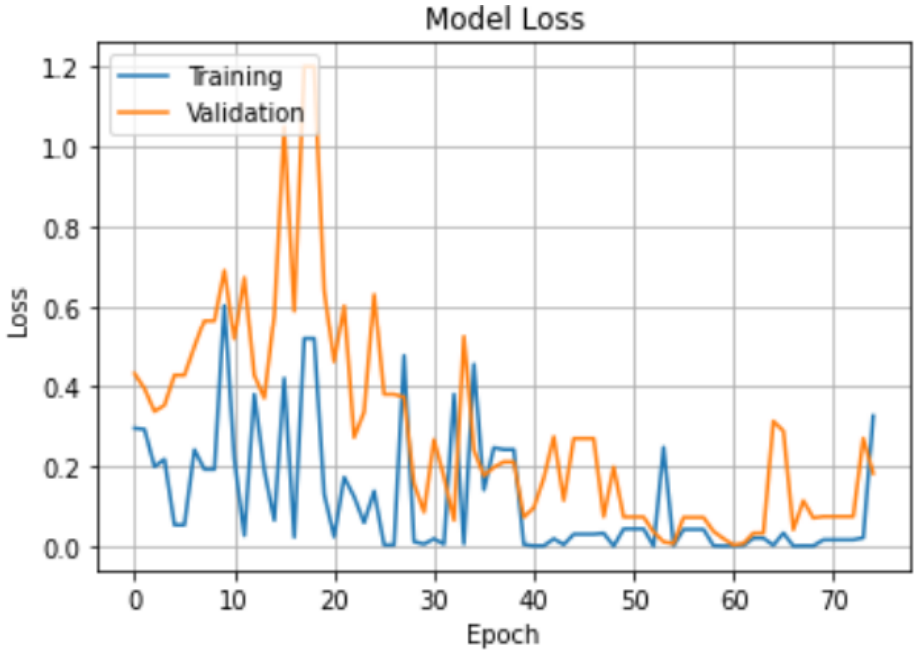


Figure 5.8: DRFM Model Loss

5.3 Conclusion

We summarize our main conclusions based on the research and body of work produced in this thesis. Our findings are based on evidence to support any claims made. The contributions are both to the fields of radar and machine learning.

5.3.1 Findings and Contributions

Findings

The main objective of the experiments was to determine if the CNN model is viable and useful. The first test objective was to test against various non-convex functions. The results in Figures 5.3 and 5.4 suggest that the ALRM method works well with closed-form non-convex functions. Convergence occurs rather quickly when compared to a standard SGD approach.

The next experiment set utilized a commonly used CIFAR-10 dataset to address any issues with data integrity and bias. This provides the ALRM method with known good data. The results in Figure 5.5 are promising and very good, achieving an overall training accuracy of 98% and testing accuracy of 87%. There is still a little overfit that needs to be addressed. Taken as a whole the results are encouraging.

The final set of experiments focused on the thesis problem of the identification of false radar targets. The results found here are good and there is overfit here as well, that can be improved upon. While the training accuracy was near perfect at 99%, the

testing accuracy was good and peaked at 80% in Figure 5.7. These results were best with smaller batch sizes which are opposite of the previous testing with CIFAR-10. The reason seems clear that we only trained with about 3,200 images with this set of experiments, while the previous set of experiments used 50,000 images.

Experiment Type	Description	Observations
Function	Non-convex function testing	• Excellent results when compared against standard SGD
CIFAR-10	Known image dataset testing	• Slight overfit but overall, very good results • Larger ResNets did not perform very well
RD	Range-Doppler image classification testing	• Smaller batches sizes worked better due to the reduced image dataset • Small overfit that can be improved upon

Table 5.4: Results Summary

Contributions

- Radar simulation - No public image data exists for radar range-Doppler images, while image dataset sets such as MNIST, CIFAR-10, and CIFAR-100 are established and very useful for model development. The MATLAB codes for the radar simulator which generates the range-Doppler images are available on a public GitHub site at `https://github.com/friedele/DRFM-Project/tree/Development` and are still in prototype development.

- CNN Model - A Jupyter notebook with detailed information on the CNN model is also available on the GitHub site, which provides the Python and MATLAB code with results from the experiments. I'm very interested in further exploration of this topic area. Plan to keep the site updated as progress is made after

the thesis delivery.

5.3.2 Future Work

- Apply to an actual real-world system. IRAD funding has been established to continue the development of these concepts and methods for ECM identification.

- Identification of false targets results were good but still need further investigation. We need to address the issue of data quality and quantity.

- Separating real targets from false targets in the radar main-lobe is a real challenge. The main-lobe of a radar beam is like a sinc function where $y(t) = \frac{sin(\pi t)}{\pi t}$ and $-\infty < t < \infty$. The detection decision is typically accomplished by hypothesis testing using a Bayes optimization which determines the optimal choice between our hypotheses. In this case, I'm considering a multi-hypothesis approach for TBD features. We could try a particle filter approach which is a recursive, Bayesian state estimator that uses discrete particles to approximate the posterior distribution of the estimated state.

- We did focus on a few hyperparameters and provided a good mathematical explanation; however, there is still much work to be done here.

Chapter 6

Appendix A - Source Code

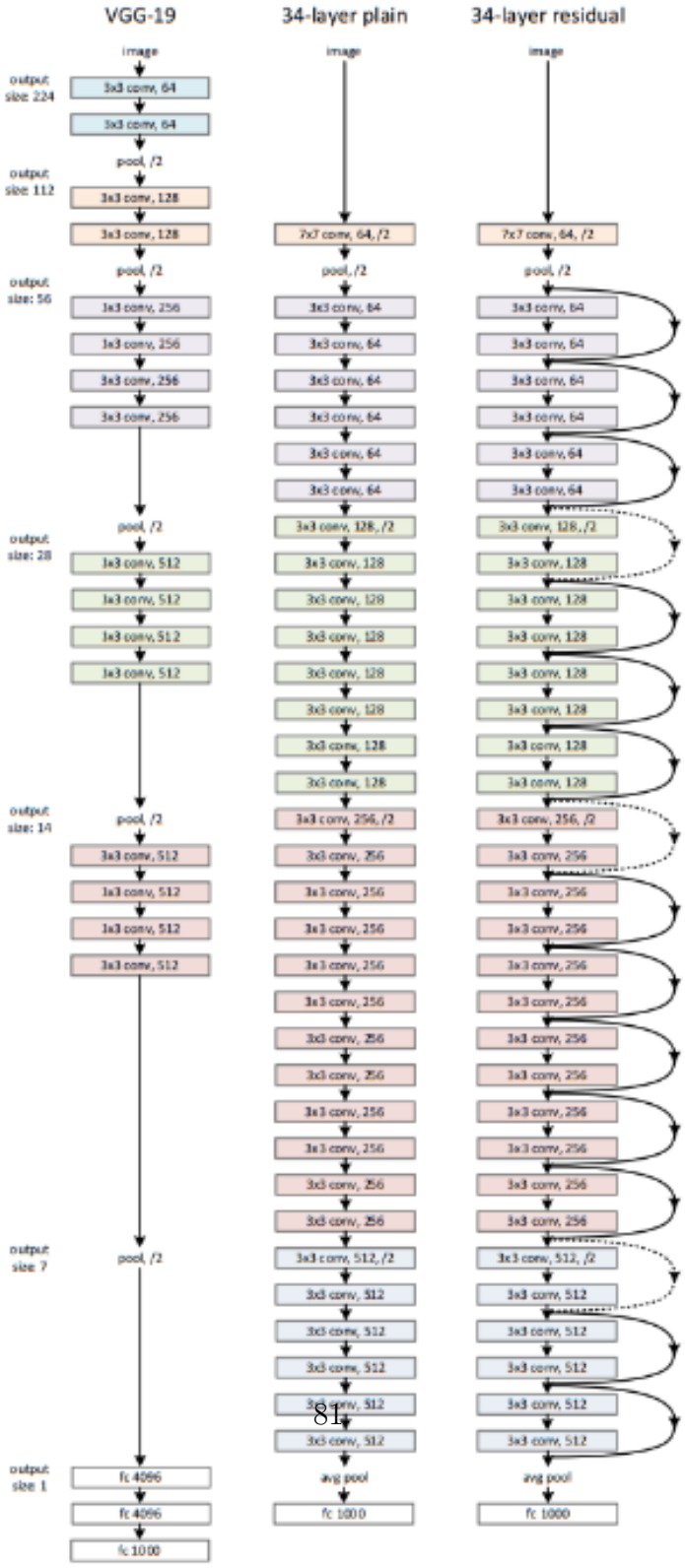
6.1 MATLAB and Python Source Code

See the public GitHub site at `https://github.com/friedele/DRFM-Project/tree/Development`

Chapter 7

Appendix B - ResNet34

Architecture



Bibliography

[1] B. Luo and L. Liu, "Development of radar active jamming recognition technology," in *2019 2nd International Conference on Mechanical Engineering (MEIMIE*, 2019, p. 462.

[2] R. J. Wiegand, "Drfm patent." [Online]. Available: https://patents.google.com/patent/US4743905A/en

[3] S. Zhao, Y. Zhou, L. Zhang, Y. Guo, and S. Tang, "Discrimination between radar targets and deception jamming in distributed multiple-radar architectures," *IET Radar, Sonar & Navigation*, vol. 11, no. 7, pp. 1124–1131, 2017.

[4] M. Greco, F. Gini, and A. Farina, "Radar detection and classification of jamming signals belonging to a cone class," *IEEE transactions on signal processing*, vol. 56, no. 5, pp. 1984–1993, 2008.

[5] M. Pak and S. Kim, "A review of deep learning in image recognition," in *2017 4th International Conference on Computer Applications and Information Processing Technology (CAIPT)*, 2017, pp. 1–3.

BIBLIOGRAPHY

[6] W. M. Walter Pitts, "A timeline of machine learning history." [Online]. Available: https://www.techtarget.com/whatis/A-Timeline-of-Machine-Learning-History

[7] Z. Hao, W. Yu, and W. Chen, "Recognition method of dense false targets jamming based on time-frequency atomic decomposition," *The Journal of Engineering*, vol. 2019, no. 20, pp. 6354–6358, 2019.

[8] D. Su and M. Gao, "Research on jamming recognition technology based on characteristic parameters," pp. 303–307, 2020.

[9] J. Lin and X. Fan, "Radar active jamming recognition based on recurrence plot and convolutional neural network," in *2021 IEEE 4th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, vol. 4, 2021, pp. 1511–1515.

[10] G. Wang, Y. Wang, S. Dong, and G. Huang, "Multiple transformation analysis for interference separation in tdcs," *Journal of Systems Engineering and Electronics*, vol. 33, no. 5, pp. 1064–1078, 2022.

[11] Q. Lv, Y. Quan, W. Feng, M. Sha, S. Dong, and M. Xing, "Radar deception jamming recognition based on weighted ensemble cnn with transfer learning," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–11, 2022.

[12] "The missile defense system." [Online]. Available: https://www.mda.mil/system/system.html

[13] M. I. Skolnik, *Introduction to Radar Systems.* McGraw-Hill, 2001, ch. 2.2, pp. 31–32.

[14] "Parsevals Theorem." [Online]. Available: https://mathworld.wolfram.com/ParsevalsTheorem.html

[15] M. A. Richards, *Fundamentals of Radar Signal Processing.* McGraw-Hill, 2005, ch. 6.1.1, pp. 297–298.

[16] I. N. Gibra, *Probability and statistical inference for scientists and engineers.* Prentice Hall, 1973.

[17] "CFAR noise jammers." [Online]. Available: https://www.researchgate.net/publication/221616765_CFAR_detectors_in_presence_of_jammer_noise

[18] "BriteCloud DRFM (Digital RF Memory) countermeasure." [Online]. Available: https://electronics.leonardo.com/en/products/britecloud-3

[19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," https://pytorch.org.

[20] L. E. Lwakatare, A. Raj, I. Crnkovic, J. Bosch, and H. H. Olsson, "Large-scale machine learning systems in real-world industrial settings: A review of

challenges and solutions," *Information and Software Technology*, vol. 127, p. 106368, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584920301373

[21] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016, p. 174.

[22] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer, 2006, vol. 4, no. 4, p. 227.

[23] R. H. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung, "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit," *nature*, vol. 405, no. 6789, pp. 947–951, 2000.

[24] A. F. Agarap, "Deep learning using rectified linear units (relu)," *CoRR*, vol. abs/1803.08375, 2018. [Online]. Available: http://arxiv.org/abs/1803.08375

[25] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," *CoRR*, vol. abs/1609.04836, 2016. [Online]. Available: http://arxiv.org/abs/1609.04836

[26] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," *Neural Networks: Tricks of the Trade: Second Edition*, pp. 437–478, 2012.

[27] Z. Hao, Y. Jiang, H. Yu, and H. Chiang, "Adaptive learning rate and momentum for training deep neural networks," *CoRR*, vol. abs/2106.11548, 2021. [Online]. Available: https://arxiv.org/abs/2106.11548

[28] S. Vaswani, A. Mishkin, I. H. Laradji, M. Schmidt, G. Gidel, and S. Lacoste-Julien, "Painless stochastic gradient: Interpolation, line-search, and convergence rates," *CoRR*, vol. abs/1905.09997, 2019. [Online]. Available: http://arxiv.org/abs/1905.09997

[29] F. Bach and E. Moulines, "Non-Asymptotic Analysis of Stochastic Approximation Algorithms for Machine Learning," in *Neural Information Processing Systems (NIPS)*, Spain, 2011, pp. –. [Online]. Available: https://hal.science/hal-00608041

[30] E. K. Ryu and S. Boyd, "Stochastic proximal iteration: a non-asymptotic improvement upon stochastic gradient descent," *Author website, early draft*, 2014.

[31] P. Toulis, D. Tran, and E. Airoldi, "Towards stability and optimality in stochastic gradient descent," in *Artificial Intelligence and Statistics*. PMLR, 2016, pp. 1290–1298.

[32] "Quasi-Newton Methods." [Online]. Available: https://optimization.cbe.cornell.edu/index.php?title=Quasi-Newton_methods

[33] R. H. Byrd, J. Nocedal, and R. B. Schnabel, "Representations of quasi-newton

matrices and their use in limited memory methods," *Mathematical Programming*, vol. 63, no. 1-3, pp. 129–156, 1994.

[34] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *Ussr computational mathematics and mathematical physics*, vol. 4, no. 5, pp. 1–17, 1964.

[35] M. Assran and M. Rabbat, "On the convergence of nesterov's accelerated gradient method in stochastic settings," *arXiv preprint arXiv:2002.12414*, 2020.

[36] C. Liu and M. Belkin, "Accelerating sgd with momentum for over-parameterized learning," *arXiv preprint arXiv:1810.13395*, 2018.

[37] J. C. Spall, *Introduction to stochastic search and optimization: estimation, simulation, and control*. John Wiley & Sons, 2005.

[38] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.