IMPROVING ULTRA-WIDEBAND LOCALIZATION BY DETECTING
RADIO MISCLASSIFICATION

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Cory Andrew Mayer

December 2018

© 2018
Cory Andrew Mayer
ALL RIGHTS RESERVED

COMMITTEE MEMBERSHIP

TITLE: Improving Ultra-Wideband Localization
by Detecting Radio Misclassification

AUTHOR: Cory Andrew Mayer

DATE SUBMITTED: December 2018

COMMITTEE CHAIR: Bruce Debruhl, Ph.D.
Professor of Computer Science

COMMITTEE MEMBER: Hugh Smith, Ph.D.
Professor of Computer Science

COMMITTEE MEMBER: John Seng, Ph.D.
Professor of Computer Science

ABSTRACT

Improving Ultra-Wideband Localization by Detecting Radio Misclassification

Cory Andrew Mayer

The Global Positioning System (GPS) and other satellite-based positioning systems are often a key component in applications requiring localization. However, accurate positioning in areas with poor GPS coverage, such as inside buildings and in dense cities, is in increasing demand for many modern applications. Fortunately, recent developments in ultra-wideband (UWB) radio technology have enabled precise positioning in places where it was not previously possible by utilizing multipath-resistant wide band pulses.

Although ultra-wideband signals are less prone to multipath interference, it is still a bottleneck as increasingly ambitious projects continue to demand higher precision. Some UWB radios include on-board detection of multipath conditions, however the implementations are usually limited to basic condition checks. In order to address these shortcomings, We propose an application of machine learning to reliably detect non-line-of-sight conditions when the on-board radio classifier fails to recognize these conditions. Our solution includes a neural network classifier that is 99.98% accurate in a variety of environments.

ACKNOWLEDGMENTS

Thanks to:

- My advisor, Dr. Debruhl for guidance and feedback during the past year
- Josh Senna and Humatics for their support of this thesis
- Camp San Luis Obispo, and the California Cybersecurity Institute, for providing access to a large warehouse for data collection
- Dr. E. Rodgers for the advice and encouragement
- Starbucks, for the many free refills I took advantage of while writing this paper

TABLE OF CONTENTS

	Page
LIST OF TABLES	ix
LIST OF FIGURES	xi
CHAPTER	
1 Introduction	1
1.1 Research Problem	1
1.2 Approach	2
1.3 Assumptions	2
1.4 Organization	3
2 Background / Related Work	4
2.1 Background	4
2.1.1 Ultra-Wideband Positioning	4
2.1.2 Machine Learning	5
2.1.2.1 Support Vector Machines	6
2.1.2.2 Neural Networks	8
2.1.2.3 Hyper-Parameter Optimization	10
2.1.2.4 K-Fold Cross Validation	12
2.2 Related Work	12
3 Classifier Design / Implementation	16
3.1 Baseline Model	16
3.1.1 Selection	16
3.1.2 Architecture	17
3.1.3 Implementation	17
3.1.4 Hyper-Parameter Optimization	17
3.2 Neural Network Models	20
3.2.1 Multilayer Perceptron	20
3.2.1.1 Architecture	20
3.2.1.2 Implementation	21
3.2.1.3 Hyper-Parameter Optimization	22

3.2.2	Convolutional Neural Network	22
3.2.2.1	Architecture	24
3.2.2.2	Implementation	24
3.2.2.3	Hyper-Parameter Optimization	25
3.2.3	Long Short-Term Memory Recurrent Neural Network .	27
3.2.3.1	Hyper-parameter Optimization	27
3.2.3.2	Architecture	29
3.2.3.3	Implementation	30
4	Experimental Design	31
4.1	Equipment	31
4.2	Setup	32
4.2.1	Controlled Environment	33
4.2.1.1	Procedure	34
4.2.1.2	Locations	35
4.2.2	Real World	37
4.2.2.1	Procedure	37
4.2.2.2	Locations	38
5	Validation / Results	43
5.1	Success Criteria	43
5.1.1	Performance Metrics	43
5.2	Hardware	44
5.3	Experimental Results	45
5.3.1	SVM (Baseline)	45
5.3.2	Multilayer Perceptron	47
5.3.3	Convolutional Neural Network	48
5.3.4	LSTM	50
5.4	Testing Results	52
5.5	Discussion	52
6	Conclusion and Future Work	55
6.1	Contributions	55
6.2	Future Work	56
	Bibliography	57

APPENDICES

A	Code Listing	65
B	Hyper-Parameter Search Results	86
C	Neural Network Cross Validation Data	90
D	Antenna Beam Patterns	119

LIST OF TABLES

Table	Page
2.1 Sample 5-Fold Cross-Validation Split	12
2.2 Summary of Related Work	15
3.1 SVM Hyper-Parameter Search Space	18
3.2 SVM Hyper-Parameter Optimization Results	19
3.3 MLP Hyper-Parameter Search Space	23
3.4 MLP Best Hyper-Parameters	23
3.5 CNN Hyper-Parameter Search Space	26
3.6 CNN Best Hyper-Parameters	27
3.7 LSTM Hyper-Parameter Search Space	28
3.8 LSTM Best Hyper-Parameters	29
5.1 Experimental Hardware Configuration	44
5.2 SVM K-Fold Cross-Validation Results	46
5.3 Performance of Deep Learning Models on New Data	52
5.4 Summary of Results	52
B.1 SVM Hyper-Parameter Optimization Results	86
B.2 MLP Hyper-Parameter Optimization Results	87
B.3 CNN Hyper-Parameter Optimization Results	87
B.4 LSTM Hyper-Parameter Optimization Results	88
C.1 MLP 5-Fold Cross Validation Fold 1	91
C.2 MLP 5-Fold Cross Validation Fold 2	94
C.3 MLP 5-Fold Cross Validation Fold 3	97
C.4 MLP 5-Fold Cross Validation Fold 4	100
C.5 MLP 5-Fold Cross Validation Fold 5	103
C.6 CNN 5-Fold Cross Validation Fold 1	106
C.7 CNN 5-Fold Cross Validation Fold 2	108
C.8 CNN 5-Fold Cross Validation Fold 3	110

C.9	CNN 5-Fold Cross Validation Fold 4	111
C.10	CNN 5-Fold Cross Validation Fold 5	113
C.11	LSTM 5-Fold Cross Validation Fold 1	115
C.12	LSTM 5-Fold Cross Validation Fold 2	115
C.13	LSTM 5-Fold Cross Validation Fold 3	116
C.14	LSTM 5-Fold Cross Validation Fold 4	117
C.15	LSTM 5-Fold Cross Validation Fold 5	117

LIST OF FIGURES

Figure	Page
2.1 2-Way Time-Of-Flight Ranging	4
2.2 Machine Learning Methods	6
2.3 Support Vector Machine Classification	7
2.4 RBF Kernel Mapping	7
2.5 Two Layer Multilayer Perceptron Architecture	8
2.6 Common Convolutional Neural Network Architecture	9
2.7 Recurrent Neural Network Unit	9
2.8 Parameters vs Hyper-Parameters for a MLP	10
3.1 Multilayer Perceptron Neural Network Architecture	21
3.2 Convolutional Network Architecture	25
3.3 Long Short Term Memory Neural Network Architecture	30
4.1 Time Domain P440 Transceiver with BroadSpec Antenna	31
4.2 Controlled Experimental Setup	33
4.3 BroadSpec™ Antenna With Phase Center Indicated	34
4.4 LOS Measurement at Grid Point	35
4.5 NLOS Measurement at Grid Point	36
4.6 Signal Blocker and Reflector Positioning	37
4.7 Camp San Luis Obispo Long Range LOS Measurements	38
4.8 Camp San Luis Obispo Long Range NLOS Measurements	39
4.9 Networks Lab Setup	40
4.10 Ranging Plot	40
4.11 Networks Lab Desktop Tower Experimental Setup	41
4.12 Maze Building Experimental Setup	41
4.13 Courtyard Experimental Setup	42
4.14 Architecture Graveyard Experimental Setup	42
4.15 Outdoor Bike Racks Experimental Setup	42

5.1	LOS Range (scaled, normalized)	45
5.2	NLOS Range (scaled, normalized)	46
5.3	Support Vector Machine Learning Curve	47
5.4	MLP Neural Network K-Fold Cross Validation Accuracy . .	47
5.5	MLP Neural Network K-Fold Cross Validation Loss	48
5.6	MLP Neural Network Training vs Validation Accuracy . . .	48
5.7	Convolutional Neural Network K-Fold Cross Validation Accuracy	49
5.8	Convolutional Neural Network K-Fold Cross Validation Loss	49
5.9	Convolutional Neural Network Training vs Validation Accuracy	50
5.10	LSTM Neural Network K-Fold Cross Validation Accuracy .	51
5.11	LSTM Neural Network K-Fold Cross Validation Loss	51
5.12	LSTM Neural Network Training vs Validation Accuracy . .	51
D.1	Time Domain Broadspec Antenna Azimuth Beam Pattern .	119
D.2	Time Domain Broadspec Antenna Elevation Beam Pattern .	120

Chapter 1

INTRODUCTION

1.1 Research Problem

Ultra-wideband (UWB) localization has become the standard approach for indoor localization due to its resistance to multipath interference commonly encountered indoors, and poor indoor GPS coverage. However, multipath interference has been found to still be a bottleneck in applications that demand high precision. Detection and mitigation of multipath signals—also called non-line-of-sight signals (NLOS)—is a well-researched problem, however to our knowledge their deployment in production UWB localization systems is limited, and there are even fewer that take advantage of the recent developments in neural networks.

One of the most advanced UWB radios in use today, the Time Domain P440 transceiver [1], only includes a primitive NLOS tagging mechanism that uses basic condition checks which can be unreliable. Our research question is to explore several neural network architectures that could be deployed onto a production UWB radio’s FPGA to augment the existing NLOS flagging and improve localization accuracy by detecting radio misclassification. Ideally, such a solution could be trained online, with the neural network model improving as the upstream localizer provides feedback on the quality of ranges in the context of other sensors. For this reason, training time, prediction time, and accuracy are all important.

1.2 Approach

In this paper, we will collect data in diverse environments and test several neural network architectures to find a good solution for this problem. The inputs for the classifiers will be the raw sampled waveforms from the radio, and the output will be whether or not the radio misclassified the waveform as a LOS range.

The neural networks will be compared against a shallow vector machine (SVM) classifier, which is a common ML classifier used for LOS/NLOS detection in ultra-wideband localization [2]. We will collect metrics for accuracy, training time, and prediction time, and evaluate the feasibility of deploying the networks in production hardware.

1.3 Assumptions

To solve this problem, we are assuming that the NLOS ranges we collect are from ranges that the ultra-wideband transceiver incorrectly classified as LOS. This assumption is made so that an accurate classifier can be developed, and the existing NLOS tagging on the radio can still be taken advantage of. Ranges that were collected are clearly LOS or NLOS so that the classifier can be trained reliably. No half-obstructed ranges were used for training or testing, and ranges were collected in stationary positions.

Additionally, for the purposes of this paper we are assuming that we have a desktop computer available with Python and GPU accelerated Tensorflow. For future work, these models could be adapted to embedded hardware such as the radio's onboard field-programmable gate array (FPGA).

1.4 Organization

In Chapter 2 we will provide background on machine learning, neural networks, and the tools that we used, as well as context for our research in existing work. Chapter 3 will document the design of our neural network classifiers and how they were implemented. Chapter 4 will show how we designed our experimental procedures and what equipment we used. In Chapter 5 we will present our findings and discuss the results. Finally, in Chapter 6 we will present an overview of our contributions and directions for future work.

Chapter 2

BACKGROUND / RELATED WORK

2.1 Background

2.1.1 Ultra-Wideband Positioning

Figure 2.1: 2-Way Time-Of-Flight Ranging

Ultra-wideband is defined as RF communication with a bandwidth of at least 500 MHz or at least 20% of the center frequency [3]. Ultra-wideband provides several beneficial characteristics for positioning applications. The wide bandwidth in the frequency domain provides a very precise resolution in the time domain, which makes it possible to distinguish the multipath components. Additionally, in the presence of interference and noisy environments, there is a high probability that some frequency components will be unaffected [4].

The use of ultra-wideband communications was largely restricted to military use until 2002, when the FCC issued a report and order allowing the unlicensed use of UWB in 3.1 to 10.6 GHz range [3]. Since then, it has been

adopted for use in positioning across a wide range of disciplines, allowing accuracy on the order of centimeters [4].

A popular technique for determining the distance between two ultra-wideband radios is called two-way time-of-flight ranging. The process is shown in Figure 2.1, and works by having one node, the requestor, send a pulse to another node, the responder. The responder determines the leading edge of the received signal, and then sends a pulse back to the requestor after a predefined period of time. Once the requestor determines the leading edge of the response, it can calculate the distance between the two radios by subtracting the responder’s predefined time delay from the total time it took to get a response pulse.

2.1.2 Machine Learning

Machine learning is a field of artificial intelligence that concerns programming computers to learn how to perform tasks based on data without being explicitly programmed to do so [5]. While shallow learning techniques, or machine learning that requires feature engineering, have been around since the 1800s, deep machine learning classifiers with the ability to learn features from large amounts of data did not become popular until 2006 when neural networks were revitalized by new algorithmic developments and increased computational power [6]–[8]. Additionally, machine learning can be divided into supervised and unsupervised learning, where supervised learning is where you provide the machine learning algorithm with labelled input to learn from, and unsupervised learning categorizes unlabeled data [9]. An overview of popular machine learning techniques are shown in Figure 2.2. For this paper we will focus on supervised deep learning techniques, and use a shallow learning model as a baseline. This section provides a brief overview of the types of machine

Figure 2.2: Machine Learning Methods. Image downloaded from <https://www.mathworks.com/discovery/machine-learning.html> on Nov 28, 2018.

learning that we will use in this paper.

2.1.2.1 Support Vector Machines

Support vector machines are shallow machine learning models that are used for classification and regression, and work by introducing hyperplanes to create boundaries between different classes in the data [10]. A simple example is using a line to separate data on a two-dimensional plot, as shown in Figure 2.3.

The linear SVM is the most basic form of support vector machine, and was introduced in 1963 [11]. A linear SVM involves separating a n dimensional

Figure 2.3: Support Vector Machine Classification. Image downloaded from https://en.wikipedia.org/wiki/Support_vector_machine#/media/File:Kernel_Machine.svg on Nov 28, 2018.

Figure 2.4: RBF Kernel Mapping. Image from Fletcher, Tristan. (2009). Support Vector Machines Explained.

vector using a $n - 1$ dimensional hyper-plane. However, if the data is not linearly separable a different solution is required. In 1992, a technique was introduced called the kernel trick, which maps input data into a higher dimensional space in order to make the data separable. A function called the kernel function is then used to operate on data which allows working with data in the higher dimensional space without having to transform the data itself [12]. The radial basis function (RBF) kernel is one of the more popular kernels used today [13], and an example of it being applied effectively to a two-dimensional dataset is shown in Figure 2.4.

2.1.2.2 Neural Networks

Neural networks are a deep learning technique based on a simplification of how the human brain works [14]. They consist of artificial neurons, also called perceptrons, that produce some output based on inputs from other neurons (and in some cases previous inputs). The first neural networks were conceived in the 1950s, but they didn't become widely used until 2006 when Geoff Hinton applied the backpropagation algorithm and modern computer hardware to the problem [8], [15]. There are many different ways to arrange neurons in a network that produce different neural network architectures, but for this paper we will use three popular architectures.

Figure 2.5: Two Layer Multilayer Perceptron Architecture. Image downloaded from https://commons.wikimedia.org/wiki/File:Artificial_neural_network.svg on Nov 28, 2018.

The most common and simple of these configurations is called a multilayer perceptron (MLP), shown in Figure 3.1. MLPs are feedforward neural networks consisting of multiple layers of fully connected neurons. Feedforward means that the data flows forward through each layer and outputs at the final layer, while fully connected means that each node is connected to every other node in the previous and next layer (but not the same layer). Each neuron computes a linear function on the input data controlled by parameters called

weights and biases which are learned during training. [16] [17].

Figure 2.6: Common Convolutional Neural Network Architecture. Image downloaded from https://en.wikipedia.org/wiki/Convolutional_neural_network#/media/File:Typical_cnn.png on Nov 28, 2018.

Convolutional neural networks (CNN) are a deep learning technique similar to MLPs, but are designed to work well on higher dimensional input such as images. Instead of fully connected neurons, each convolutional layer contains filters which produce activations based on data in a sliding window fashion. In this way, local patterns in the data can be recognized by the convolutional neural network. In this type of architecture the number of filters, filter size, as well as how far to slide the filter for each activation are hyper-parameters (see subsection 2.1.2.3) [17] [18]. An example of a typical convolutional neural network is shown in Figure 2.6.

Figure 2.7: Recurrent Neural Network Unit. Image downloaded from https://en.wikipedia.org/wiki/Recurrent_neural_network#/media/File:Recurrent_neural_network_unfold.svg (modified) on Nov 28, 2018.

Long short-term memory (LSTM) deep neural networks are a type of recurrent neural network (RNN) which are stateful, and thus use information

about past data points to influence the classification of new data points [19]. A simple RNN unit is pictured in Figure 2.7. Traditional RNNs suffered from a problem known as the exploding gradient, in which the loops in data from nodes to themselves would cause the gradient to grow unwieldy large or small, which would inhibit the ability of those nodes to update [20]. A solution was found when researchers developed LSTM, which introduced internal node memory that functions like an electronic latch, and allows nodes to retain information over a longer period of time and solves the exploding gradient problem [21].

2.1.2.3 Hyper-Parameter Optimization

Figure 2.8: Parameters vs Hyper-Parameters for a MLP. Image downloaded from <https://mljar.com/blog/are-hyper-parameters-really-important-in-machine-learning/> on Nov 28, 2018.

Machine learning algorithms often have many hyper-parameters that affect the model’s performance. Where parameters of a network control the output directly (such as weights, biases for a multilayer perceptron network), hyper-parameters affect the assignment of parameters (see Figure 2.8) [22]. The

optimization problem can be defined as:

$$\arg \min_{x \in \mathbb{R}} f(x)$$

where $f(x)$ is the objective function of the machine learning algorithm and x is a set of hyper-parameters. These hyper-parameters can be manually selected, chosen by using an optimization algorithm, or a combination of the two.

Beyond hand selection, the most basic of these algorithms is grid search. A grid search works by selecting a set of values to try for each hyper-parameter, and iterating over each possible hyper-parameter combination in an exhaustive search [23]. This type of search works well for machine learning models that can be trained relatively quickly such as linear regression and support vector machines. As models get more complex, as is the case in deep neural networks, an exhaustive search becomes less practical because of the time required to train a deep neural network [24].

To make automated hyper-parameter selection viable for DNNs, various other techniques were developed. It was found that simply randomly choosing sets of hyper-parameters, known as random search, was more efficient than grid search for neural networks [25]. However, random search is still not very efficient because it ignores feedback from past hyper-parameter choices. Sequential Model-Based Optimization (SMBO) is a class of hyper-parameter optimization where the cost function is estimated instead of explicitly computed, and a technique is used to intelligently search hyper-parameters based on past feedback. One of the most popular techniques used is SMBO with the Tree Parzen Estimator (TPE), which optimizes hyper-parameters by using a probability model of the objective function, allowing a quick estimation of the next set of hyper-parameters to try for each step [26], [27].

2.1.2.4 K-Fold Cross Validation

K-fold cross validation is considered by many to be the gold standard of machine learning algorithm evaluation [28]. It consists of splitting the dataset into K subsets, and training the model K times with one subset at a time being withheld for validation [29]. One iteration of a K-fold cross validation is shown in Table 2.1. After validating the model on all k folds, an average of the performance metric is used to evaluate the model.

Data Set				
Validation	Train	Train	Train	Train

Table 2.1: Sample 5-Fold Cross-Validation Split

2.2 Related Work

Many different methods have been researched in order to reduce NLOS error, ranging from simple least-squares estimations, to statistical analysis, and more complex machine learning algorithms. They can be separated into two types: NLOS classification, and NLOS mitigation. An overview of the most popular methods can be found in [30]. Since our paper focuses on machine learning classification of misclassified NLOS signals, we will focus on the related work in using machine learning models, which is summarized in Table 2.2.

[2], [31]–[35] all show success using an SVM classifiers with features extracted from the received waveforms. However, all of them use less than 1500 samples. It is unclear how they would perform as the size of the data set increased.

Other work suggests improvements to the use of SVM. [36] uses one-class classification to detect NLOS outliers when only LOS or NLOS data is avail-

able. [37] uses k-nearest-neighbors and presents similar results to SVM regression that is less computationally complex. [38] presents a Relevance Vector Machine (RVM) based method, which builds on SVM to improve accuracy and complexity. However, since Microsoft holds a patent on RVM [39], adoption in the community has been limited to educational use. [40] uses an Import Vector Machine (IVM), which improves the time complexity of SVM and RVM from $O(N_{\text{samples}}^3)$ to (N_{samples}^2) . Additionally they cite increased accuracy and automatic features selection as benefits. As of the time of writing, IVMs were not available in any common machine learning libraries.

We found that there is little existing work on the application of neural networks to the NLOS classification problem. Of these works, [41] is the most similar to our work. The authors build a 2-layer MLP neural network to classify “undetected direct path (UDP)”, which is when the radio used classifies the signal as direct path, but it is actually not a direct path signal. This is the same classification that our paper is focused on, however their classifier takes channel statistics as inputs instead of received waveform samples. We believe that received waveform samples are a better fit for neural networks, since neural networks are characterized by their ability to learn their own features.

To our knowledge, the only other work that uses received waveform samples as inputs (as opposed to extracting features from them), is [42]. They use a 4-layer MLP neural network, however they perform some preprocessing on the received waveform samples that interpolated and aligned the samples to the leading edge. This adds computational complexity that would make real time predictions less effective. Additionally, their work focuses on bias estimating instead of classification.

Ref.	Architecture	Inputs	Outputs	Test Environment
[41]	2-layer MLP	Channel statistics from radio and computed features.	Binary classification.	Indoor academic building. Metallic shaft present for misclassified LOS.
[32]	Least Squares SVM	Features extracted from waveforms.	Binary classification. Ranging bias also calculated.	MIT campus, indoor office environment. 1024 measurements: 512 LOS, 512 NLOS. Data used by a number of other papers.
[33]	SVM, Gaussian Process	Features extracted from waveforms.	Bias estimation.	Data from [32]
[2]	SVM	Features extracted from waveforms.	4 types of obstruction material, multiclass classification.	Indoors, 800 samples. 200 per type of materials.
[34]	LS-SVM	Features extracted from waveforms.	Binary classification. Ranging bias also calculated.	Chinese town buildings. 1000 measurements: 500 LOS, 500 NLOS. Time Domain P410 radios.
[35]	SVM	Features extracted from waveforms.	Binary classification.	Indoor single room, data collected by Antenna & Electromagnetics Group (Body WiSeR).
[36]	SVDD	Features extracted from waveforms.	Binary classification.	Data from [32].
[38]	RVM	Features extracted from waveforms.	Binary classifier and bias regressor.	200 LOS/200 NLOS training, 10^5 LOS/ 10^5 NLOS testing
[37]	k-nearest neighbor regression	Features extracted from waveforms.	Bias estimate.	Indoors, grid of measurement points with concrete blockage for NLOS. Time Domain radios used.
[43]	MLP, 3 hidden layers	Features extracted from waveforms.	Binary classification.	Office space, grid of 36 locations, 100 samples at each location. Total of 3600 samples. DecaWave DW1000 radios.

Ref.	Architecture	Inputs	Outputs	Test Environment
[31]	LS-SVM	Features extracted from simulated waveforms.	Bias estimation.	Outdoors, simulated with ray tracing and real LIDAR data. 10,000 samples from a 6x6 grid.
[42]	MLP, 3 hidden layers	Interpolated and leading edge aligned channel impulse response	Bias estimate	DecaWave DW1000 radios.
[40]	IVM	Features extracted from waveforms, automatically selected using IVM.	Binary classification.	500 LOS/500 NLOS samples generated from channel model.

Table 2.2: Summary of Related Work

Chapter 3

CLASSIFIER DESIGN / IMPLEMENTATION

In this section we will describe the design of our baseline model, as well as the three neural network architectures that we chose to evaluate for the incorrect LOS classification problem. At a high level, the model we will use to solve this problem can be defined as

$$Y = f(X)$$

where $Y \in \{0, 1\}$
and $X \in R^{1632}$

and each model will define its own $f(x)$. For each model we will describe the architecture, choice of hyper-parameters, and implementation details. The data used for input was collected as described in chapter 4, and was randomly shuffled.

3.1 Baseline Model

3.1.1 Selection

To evaluate the performance of the neural network architectures, we established a baseline machine learning model to compare them to. A support vector machine (SVM) was chosen as it is commonly used for LOS/NLOS detection for ultra-wideband ranging [2], [32]–[35], [44], [45]. First, a linear SVM (see subsection 2.1.2.1) was tested on the dataset, however we were unable to achieve convergence—the achievement of an acceptable accuracy as

determined by the hyper-parameter *tol*—in a reasonable amount of iterations. This led us to try using a nonlinear SVM with a radial basis function (RBF), since it is one of the most popular nonlinear kernels used for SVM [13]. Initial results for SVM with RBF met our success criteria (see chapter 5), so the model was selected as the baseline.

3.1.2 Architecture

The inputs to the SVM are the samples from a full received scan from the Time Domain radio. Each range contains 1632 samples. We scaled the samples to zero mean and unit variance by subtracting the mean of all collected samples from each sample, then dividing each sample by the standard deviation of all collected samples. The output was a binary classification of whether or not the ultra-wideband range was misclassified as LOS.

3.1.3 Implementation

We used the scikit-learn library [46] as a frontend to the LibSVM library [47] for training and evaluation. We additionally used scikit-learn’s **StratifiedKfold** cross validation class for k-fold cross validation (see subsection 2.1.2.4), and **RandomizedSearchCV** for the hyper-parameter search. The full code can be found in Appendix A.

3.1.4 Hyper-Parameter Optimization

We were unable to do an exhaustive search for C and γ , the hyper-parameters of an SVM with RBF kernel (see Table 3.1). This is because the nonlinear SVM has a worse case time complexity of $O(n_{\text{features}} * n_{\text{samples}}^3)$ [10], so an exhaustive search would be prohibitively time intensive. Instead we used a random search

(see subsection 2.1.2.3) because SMBO hyper-parameter optimization was not readily available for scikit-learn. The range of parameter values searched is shown in Table 3.1, and we selected them based on the recommendations from the scikit-learn documentation for SVMs with the RBF kernel [48].

Parameter	Description	Values
C	penalty parameter of error term	[1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02, 1.e+03]
γ	RBF kernel coefficient	[1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02, 1.e+03]

Table 3.1: SVM Hyper-Parameter Search Space

We performed 20 iterations of cross validated random parameter search, after which we found a set of hyper-parameter values that produced satisfactory k-fold cross validation accuracy (see subsection 2.1.2.4). The parameter search took 1773 minutes on a 14-core Intel Xeon processor at 2.3 GHz with hyper-threading enabled and 32GB of RAM. The results are shown below in Table 3.2. We found that the best parameters were $C = 1000$ and $\gamma = 0.001$. Although these parameters were the maximum C and minimum γ , they met our success criteria with a cross-validation accuracy of over 95%, so we used them for all further training.

Rank	Mean CV Score	STD CV Score	C	γ
1	0.9997	9.4245×10^{-5}	1000.0	0.001
2	0.9995	0.0002	100.0	0.01
3	0.9995	0.0003	1.0	0.01
4	0.9993	0.0002	10.0	0.001
5	0.9687	0.0010	10.0	0.1
5	0.9687	0.0010	1000.0	0.1
7	0.9335	0.0047	0.01	0.001
8	0.7764	0.0037	0.01	0.1
9	0.7614	0.0029	1.0	1.0
10	0.5001	1.3898×10^{-5}	0.1	10.0
10	0.5001	1.3898×10^{-5}	0.001	1.0
10	0.5001	1.3898×10^{-5}	100.0	100.0
10	0.5001	1.3898×10^{-5}	10.0	100.0
10	0.5001	1.3898×10^{-5}	10.0	1000.0
10	0.5001	1.3898×10^{-5}	1.0	10.0
10	0.5001	1.3898×10^{-5}	0.01	100.0
10	0.5001	1.3898×10^{-5}	1000.0	10.0
10	0.5001	1.3898×10^{-5}	0.001	1000.0
10	0.5001	1.3898×10^{-5}	1000.0	1000.0
10	0.5001	1.3898×10^{-5}	0.01	1.0

Table 3.2: SVM Hyper-Parameter Optimization Results

3.2 Neural Network Models

We chose to evaluate three neural network architectures: a multilayer perceptron, convolutional neural network, and long short-term memory neural network, because of their applicability to similar problems in literature and industry (see subsection 2.1.2.2). In this section we will describe the design of each model, why it was chosen, and the selection of hyper-parameters.

3.2.1 Multilayer Perceptron

The first neural network architecture type we chose to evaluate on our dataset was a multilayer perceptron neural network (see subsection 2.1.2.2). We chose multilayer perceptron because it has a simple architecture, and is one of the neural network architectures most used in industry today [49].

3.2.1.1 Architecture

We chose to start by selecting a neural network with a single hidden layer, since according to the Universal Approximation Theorem [50], a MLP with just one hidden layer can approximate any function. After initial testing, we found that a single hidden layer was sufficient for our application, and adding more layers did not improve the accuracy significantly.

The resulting neural network (shown in Figure 3.1) is a 2-layer multilayer perceptron network, with the input layer consisting of 1632 neurons corresponding to the 1632 signal samples for each range from the Time Domain radio. We scaled the input features using the same process that we used for the SVM classifier. The hidden layer contains 2229 neurons as determined by the hyper-parameter optimization. Finally, the output layer contains a sigmoid

Figure 3.1: Multilayer Perceptron Neural Network Architecture

activation function that predicts the binary state of the signal: misclassified as LOS or not. Dropout layers are introduced between the inner layers in order to aid with regularization, as determined by the hyper-parameter optimization.

3.2.1.2 Implementation

We implemented the MLP using the Keras deep learning framework [51]. We chose Keras because it provides a high level interface to popular high performance frameworks such as Tensorflow, and allows for rapid experimentation of new neural network architectures while taking advantage of modern hard-

ware resources such as GPU acceleration. The **Sequential** model was used to construct the MLP network. The full code listing can be found in Appendix A.

We implemented hyper-parameter optimization using the hyperas library [52], a Keras wrapper for the popular hyperopt hyper-parameter optimization library [53]. Hyperopt provides a SMBO optimizer with TPE, which is utilized via the hyperas wrapper.

3.2.1.3 Hyper-Parameter Optimization

Since there were many more hyper-parameters for the neural network models than the two for SVM with RBF, we used Sequential Model-Based Optimization with the Tree Parzen Estimator to optimize the hyper-parameters. We chose the hyper-parameter search space based on our experience and the recommendations from *Deep Learning with Python* by Francois Chollet [17], which is shown in Table 3.3.

We ran the hyper-parameter search for 30 iterations, using a test set to evaluate the performance. The best parameters produced a test set accuracy of 99.9965%, which we then evaluated using k-fold cross validation to ensure the hyper-parameters were not biased towards the test set. The search took 393 minutes. The best parameters are shown in Table 3.4. The full hyper-parameter search results are shown in Appendix B.

3.2.2 Convolutional Neural Network

The next neural network architecture we tested was a 1-D convolutional neural network (see subsection 2.1.2.2). Convolutional neural networks are most often known for their use 2-D image processing, but are also frequently used in general pattern recognition and signal processing [54]. Due to the

Parameter	Description	Values
<code>neurons_hidden</code>	# of neurons in the hidden layer	[408..3263]
<code>batch_size</code>	batch size for training	[32, 64, 128]
<code>optimizer</code>	the optimizer to use for training	['adam', 'sgd', 'rmsprop']
<code>learning_rate</code>	learning rate of the optimizer	[1e-06, 1e-05, 1e-04, 1e-03, 1e-02, 1e-01]
<code>epochs</code>	the number of training iterations	[25, 50, 100]
<code>use_regularization</code>	if the inner layers should use a kernel regularizer	[true, false]
<code>reg_lambda</code>	the λ parameter for regularization	[1e-05, 1e-04, 1e-03, 1e-02, 1e-01]
<code>inner_activation</code>	the activation function for the inner layers	['relu', LeakyReLU()]
<code>dropout</code>	the amount of dropout for the inner layers	[0,0.5]
<code>constr</code>	the kernel constraint for the inner layers	[None, maxnorm(3), maxnorm(4), maxnorm(5)]

Table 3.3: MLP Hyper-Parameter Search Space

Parameter	Value
<code>neurons_hidden</code>	2229
<code>batch_size</code>	32
<code>optimizer</code>	'adam'
<code>learning_rate</code>	1e-05
<code>epochs</code>	100
<code>use_regularization</code>	true
<code>reg_lambda</code>	1e-05
<code>inner_activation</code>	'relu'
<code>dropout</code>	0.3013
<code>constr</code>	None

Table 3.4: MLP Best Hyper-Parameters

success of CNNs in speech recognition and time-series data [55], we decided to use a 1-D convolutional neural network to detect patterns in our time-series ultra-wideband waveforms. We constructed a typical 1-D convolutional neural network to test this idea.

3.2.2.1 Architecture

As shown in Figure 3.2, we chose a simple 1-D convolutional neural network architecture similar to the one in [54]. It consists of two 1-D convolutional layers, and max pooling layers after each convolutional layer. The first convolutional layer was the input layer, and we scaled the input features in the same way they were scaled for the SVM and MLP. The LeakyReLU activation function was used for the internal layers, as it was found by the hyper-parameter optimization to be more effective than a regular ReLU. Finally, a flatten layer leads to a single neuron dense layer, which is activated by a sigmoid function to produce the binary classification.

Regularization was not introduced, because we found that in the initial experimentation the network without regularization was already performing above 95% accuracy, even without hyper-parameter optimization. We decided to try hyper-parameter optimization without adding regularization, which turned out to meet our success criteria (see chapter 5).

3.2.2.2 Implementation

Keras provides convolutional layers for the `Sequential` model, so we used it to build our convolutional neural network as well. Similar to the MLP, hyperas was used for SMBO with TPE hyper-parameter optimization.

Figure 3.2: Convolutional Network Architecture

3.2.2.3 Hyper-Parameter Optimization

The hyper-parameter search space we explored was again based on the recommendations from *Deep Learning with Python* [17]. The ranges we used were based on values used for a working model for a similar application [54], and are shown in Table 3.5.

As with the MLP, we used SBMO with TPE to optimize these parameters. After 30 iterations, we found a good set of hyper-parameters that resulted in a validation set accuracy of 99.986%. The search took 458 minutes. We used k-fold cross validation to validate the hyper-parameters and ensure there

Parameter	Description	Values
<code>kernel_size</code>	# the dimension of the 1-D convolutional kernel	[3, 4, 5, 6, 7, 8]
<code>strides_first</code>	the stride size for the first convolutional layer	[1, 2, 3, 4, 5]
<code>strides_second</code>	the stride size for the second convolutional layer	[1, 2, 3, 4]
<code>strides_pool</code>	the stride size for the pooling layer	[1, 2, 3, 4, 5, 6, 7, 8]
<code>neurons_first</code>	the # of neurons in the first convolutional layer	[32, 64, 128]
<code>neurons_second</code>	the # of neurons in the second convolutional layer	[8, 16, 32]
<code>batch_size</code>	batch size for training	[16, 32, 64]
<code>optimizer</code>	the optimizer to use for training	['adam', 'sgd', 'rmsprop']
<code>learning_rate</code>	learning rate of the optimizer	[1e-06, 1e-05, 1e-04, 1e-03, 1e-02, 1e-01]
<code>epochs</code>	the number of training iterations	[25, 50, 100]
<code>inner_activation</code>	the activation function for the inner layers	['relu', LeakyReLU()]

Table 3.5: CNN Hyper-Parameter Search Space

was no significant bias to the testing set. The best parameters are shown in Table 3.6.

Parameter	Value
<code>kernel_size</code>	7
<code>strides_first</code>	2
<code>strides_second</code>	1
<code>strides_pool</code>	8
<code>neurons_first</code>	64
<code>neurons_second</code>	8
<code>batch_size</code>	32
<code>optimizer</code>	'adam'
<code>learning_rate</code>	1e-04
<code>epochs</code>	50
<code>inner_activation</code>	LeakyReLU()

Table 3.6: CNN Best Hyper-Parameters

3.2.3 Long Short-Term Memory Recurrent Neural Network

The next model that we chose to evaluate for our application was the LSTM, which today is one of the most popular neural network architectures used in industry for applications requiring a model with state [49], [56]. We chose to use this architecture due to the similarity of UWB signal classification to other waveform pattern recognition applications that LSTM has been proven effective for [57].

3.2.3.1 Hyper-parameter Optimization

As with the other neural network architectures, we used SMBO with TPE to optimize the hyper-parameters. We chose a range of hyper-parameters based on the recommendations from *Deep Learning with Python* for a basic application of LSTM [17]. These hyper-parameter ranges are shown in Table 3.7.

After 30 iterations of the search, we found a good set of hyper-parameters

Parameter	Description	Values
<code>neurons_hidden</code>	# of neurons in the hidden layer	<code>([32, 64, 128]</code>
<code>batch_size</code>	batch size for training	<code>[8, 16, 32]</code>
<code>optimizer</code>	the optimizer to use for training	<code>'rmsprop'</code>
<code>learning_rate</code>	learning rate of the optimizer	<code>[1e-06, 1e-05, 1e-04, 1e-03, 1e-02, 1e-01]</code>
<code>epochs</code>	the number of training iterations	<code>[10, 15, 20, 40]</code>
<code>use_regularization</code>	if the inner layers should use a kernel regularizer	<code>[true, false]</code>
<code>reg_lambda</code>	the λ parameter for regularization	<code>[1e-05, 1e-04, 1e-03, 1e-02, 1e-01]</code>
<code>inner_activation</code>	the activation function for the inner layers	<code>['relu', LeakyReLU()]</code>
<code>dropout</code>	the amount of dropout for the inner layers	<code>[0,0.5]</code>
<code>constr</code>	the kernel constraint for the inner layers	<code>[None, maxnorm(3), maxnorm(4), maxnorm(5)]</code>
<code>lstm_output</code>	the output dimension of the LSTM layer	<code>[64, 128, 256]</code>

Table 3.7: LSTM Hyper-Parameter Search Space

that yielded a validation set accuracy of 100%. The search took 3,795 minutes. K-fold cross validation was performed to ensure there was not a high bias towards the testing test. The resulting hyper-parameters are shown in Table 3.8.

Parameter	Value
<code>neurons_hidden</code>	32
<code>batch_size</code>	32
<code>optimizer</code>	'rmsprop'
<code>learning_rate</code>	1e-03
<code>epochs</code>	15
<code>use_regularization</code>	true
<code>reg_lambda</code>	1e-04
<code>inner_activation</code>	LeakyReLU()
<code>dropout</code>	0.1604
<code>constr</code>	maxnorm(5)
<code>lstm_output</code>	256

Table 3.8: LSTM Best Hyper-Parameters

3.2.3.2 Architecture

Similar to MLP, there is evidence to support that even a very simple LSTM architecture is sufficient for most applications [58]. As shown in Figure 3.3, we constructed a simple LSTM network with one LSTM layer, one hidden dense layer, and an output dense layer activated by a sigmoid function for the binary classification. The input to the LSTM layer is a 1632 dimensional vector consisting of the 1632 samples per waveform, scaled and normalized in the same way as the SVM, MLP, and CNN.

As selected by the hyper-parameter optimization, we activated the internal layers with a LeakyReLU function, and inserted a dropout between layers of 0.1604. The LSTM layer has 256 hidden units, and the hidden dense layer has 32 units. A maxnorm(5) kernel constraint was used for the hidden dense

layer.

Figure 3.3: Long Short Term Memory Neural Network Architecture

3.2.3.3 Implementation

Keras also provides an efficient implementation of the LSTM layer via the CuDNNLSTM layer. This layer takes advantage of CUDA on the GPU to optimize the LSTM implementation. Similar to the other neural networks in Keras, hyperas was used for the SMBO with TPE hyper-parameter optimization.

Chapter 4

EXPERIMENTAL DESIGN

4.1 Equipment

We used two Time Domain PulsON[®] 440 transceivers (pictured in Figure 4.1) to collect ultra-wideband ranges, range statistics, and sampled waveforms. These ultra-wideband radios operate with a bandwidth of 3.1 to 4.8 GHz, and use two-way time of flight to determine the distance between two radios (see chapter 2). They are capable of providing ranges at a sustained rate of 125 Hz with <2 cm standard deviation accuracy [1]. The radios were fitted with Time Domain Broadspec ultra-wideband antennas [59]. The beam patterns for this antenna can be found in Appendix D.

Figure 4.1: Time Domain P440 Transceiver with Broadspec Antenna

We mounted the radios on camera tripods with smartphone adapters. This allowed us to secure the radios, easily reposition them, and set a consistent height. The mounting hardware can be seen in Figure 4.1. We powered the ra-

dios with 5V portable battery packs via micro-USB, which we placed below the antenna where they would likely cause minimal interference with transmission.

For the experiments, we connected the radios via 10/100 ethernet to a laptop with a 2.9 GHz Intel Core i7 quad-core processor with 16 GB of memory running Microsoft Windows 10. We recorded range measurements, range statistics, and sampled waveforms from the radios using the Time Domain RangeNet software, which is capable of logging to CSV [60]. We configured the radios for ranging mode, and enabled logging of full ranges and scans. Ranges were collected at a rate of 40 Hz, which we experimentally found to be sustainable over ethernet with detailed logging enabled. For each range, 1632 samples were recorded with int32 precision in 61 ps intervals, starting 90 ns before the leading edge and ending 10 ns after the leading edge.

4.2 Setup

We designed the experiment to maximize the accuracy and generalization of the classifier. Since the classifier needed to detect when a received range was non-line-of-sight when the radio reported that it was line-of-sight, we designed the initial experiment to be a more controlled setup. Radios were in fixed positions and an apparatus was introduced to change the LOS/NLOS state such that the radio would still detect that the ranges were LOS, even though the range was NLOS.

After some initial training of the machine learning classifiers, we introduced some data that was collected without the LOS/NLOS apparatus and found that it did not generalize well. We then decided to collect more data with a setup more similar to the environment that would be found in a real world application of ultra-wideband localization. The new experiment removed the

LOS/NLOS apparatus and was set up outdoors where we were able to reproduce the research problem and without prescribed distances between radios. The description of these two versions of the experimental setup used to collect data for the classifier are described below.

4.2.1 Controlled Environment

Figure 4.2: Controlled Experimental Setup

4.2.1.1 Procedure

As shown in Figure 4.2, we placed one UWB transceiver in a fixed position, which we will refer to from now on as the *anchor node*. We moved around the other transceiver, which we will refer to as the *mobile node*, in a grid formation in order to collect ranges from different distances and angles between the radios. A large tarp was marked with the grid points to aid in data collection.

Ranges were initiated from the anchor node, and the received response from the mobile node was logged by the anchor node. Both radios were mounted at a vertical height of 8 feet from the ground to the antenna's phase center (see Figure 4.3), which is the point on the antenna where the range is measured from [59].

Figure 4.3: BroadSpec™ Antenna With Phase Center Indicated. Image from [59].

For each point on the grid, we collected 500 ranges with a clear line of sight between the radios. Then, two devices were introduced to obstruct the direct path between the radios and create a path for a clear reflection to trick the radios into tagging the ranges as line-of-sight. At this point we collected 500 ranges with the non-line-of-sight configuration. An example of the LOS configuration is shown in Figure 4.4, and the NLOS configuration is shown in Figure 4.5.

The signal blocker was a 4x8 foot piece of particle board covered with 4

Figure 4.4: LOS Measurement at Grid Point

layers of aluminum foil. The reflector we used was a 1x1 foot 24 gauge copper plate. We placed the signal blocker within a 3 foot radius of the mobile node, in a location that was optimal to block the signal and cause the radio to lock on to reflected ranges (this varied per point). For each point, once we determined the optimal blocked location, the reflector was placed 3 feet away, perpendicular to the signal blocker and rotated 90 degrees (see Figure 4.5, Figure 4.6).

Ranges collected in the LOS configuration were given the label one, while ranges collected with the NLOS configuration were given the label zero. With a 5x5 grid of points, we collected a total of 25,000 range measurements per location.

4.2.1.2 Locations

We chose 2 indoor locations for the controlled setup. Large indoor areas were chosen because the environment was easier to control, and indoors is where ultra-wideband is commonly used.

Figure 4.5: NLOS Measurement at Grid Point

Large Warehouse The first location (shown in Figure 4.4) we used was a large warehouse at Camp San Luis Obispo, which access to was granted by the California Cybersecurity Institute. The area was approximately 50,000 square feet, and was a concrete structure with a metal roof. We followed the standard procedure at this location

Since there was a large area available, 6000 additional ranges were collected (3000 LOS, 3000 NLOS) at a longer distance of 70 feet in a straight line. The LOS and NLOS setups are shown in Figure 4.7 and Figure 4.8 respectively. At this range it was difficult to trick the radio’s NLOS detection, but we achieved it by modifying the procedure to position the signal blocker 1’10” from the anchor node, and the reflector 2’7” from the signal blocker, 1’ off the perpendicular line.

Networks Lab The other location we chose was a laboratory at Cal Poly (shown in Figure 4.9). We chose it because it contained many metallic objects which the ranges could bounce off of. We carried out the standard procedure at this location.

Figure 4.6: Signal Blocker and Reflector Positioning

4.2.2 Real World

4.2.2.1 Procedure

For this experiment, we relaxed some of the constraints to better model a real world application of ultra-wideband localization. Specifically, we visited five locations on the Cal Poly campus and moved around both the anchor node and mobile node until we found a positioning that was NLOS but the radios detected as LOS. We made this determination by observing when ranges flagged as LOS suddenly jumped by a distance much larger than the radio was moved, using the Time Domain RangeNet software [60].

An example of this happening is shown in Figure 4.10. As we were moving the radio back and forth approximately 6 inches, the range measurements jumped by more than 3 feet. In this example the ranges greater than 2500 mm could be confidently labeled as NLOS. We chose four locations to include in the experimental data, and a fifth location for final testing on the trained models.

Figure 4.7: Camp San Luis Obispo Long Range LOS Measurements

4.2.2.2 Locations

Networks Lab Desktop Tower Figure 4.11 shows the first location chosen for a more realistic setup. We used stack of old desktop computers to block the direct path, and observed a reflection that was tagged by the radio as LOS. We collected 3000 LOS ranges and 3000 NLOS ranges, and these were included in the experimental data.

Architecture “Maze” Building Figure 4.12 shows the placement of the anchor and mobile nodes for a misclassified LOS location found at the Maze Building. We suspect the NLOS ranges came from the reflection from the nearby bicycles or metal walls. We collected 1000 NLOS ranges and 1000 LOS ranges which were included in our experimental data.

Engineering West Courtyard Figure 4.13 shows the experimental setup for a misclassified NLOS location found at the Engineering West Courtyard. It is likely that the NLOS ranges were reflected off of the large metallic structure pictured. We collected 1000 NLOS ranges and 1000 LOS ranges, which were

Figure 4.8: Camp San Luis Obispo Long Range NLOS Measurements

included in our experimental data.

Architecture “Graveyard” Design Village Figure 4.14 shows the experimental setup for a misclassified NLOS location found at a structure in the Cal Poly Architecture Design Village, also known as the Architecture Graveyard. We chose this location because we suspected that the large metallic structure would provide multiple paths for the range to reflect off of if the direct path was blocked. Once the path was blocked, we did indeed easily trick the radio into classifying NLOS ranges as LOS. We collected 1000 NLOS ranges and 1000 LOS ranges, and they were included in our experimental data.

Outdoor Bike Racks Figure 4.15 shows the experimental setup for our final location, used for testing of the models once they had already been trained. This location is outdoors near a large array of metal bike racks. When the mobile node was moved to the right to be blocked by the side of the building, we observed an incorrect LOS classification by the ultra-wideband radios. We collected 1000 NLOS ranges and 1000 LOS ranges.

Figure 4.9: Networks Lab Setup

Figure 4.10: Ranging Plot

(a) LOS

(b) NLOS

Figure 4.11: Networks Lab Desktop Tower Experimental Setup

(a) LOS

(b) NLOS

Figure 4.12: Maze Building Experimental Setup

(a) LOS

(b) NLOS

Figure 4.13: Courtyard Experimental Setup

(a) LOS

(b) NLOS

Figure 4.14: Architecture Graveyard Experimental Setup

(a) LOS

(b) NLOS

Figure 4.15: Outdoor Bike Racks Experimental Setup

Chapter 5

VALIDATION / RESULTS

The following chapter presents the selection and evaluation of the machine learning models as well as the experimental results.

5.1 Success Criteria

5.1.1 Performance Metrics

Depending on the dataset, different measures of performance for a neural network can be useful. Some of the most common include accuracy, precision, recall, and the f1 score.

Accuracy is a measurement of how many samples were classified accurately. It is the most intuitive metric, but may be problematic when there is not a good balance of positive and negative examples. For example, consider a binary classifier with only 1% positive samples. Then a 99% classification accuracy may make it seem like the algorithm performed well, but it could accomplish this by always predicting negative. In this case accuracy does not give us a good indication of the performance.

A metric that can provide more information than accuracy is the f1 score. At a high level, it measures the balance between precision and recall, where precision is defined as $\frac{\text{true positives}}{\text{true positive} + \text{false positives}}$, and recall is defined as $\frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$ [61]. A high f1 score indicates that there are not too many false positives or false negatives.

We use accuracy as our main metric, because our dataset consists of ap-

proximately half positives and half negatives. Thus, accuracy is a reliable metric for our classifier’s performance. For the purposes of our experiments, we will consider a k-fold cross-validation accuracy of over 95% as successful. We will also include f1 scores for comparison.

Additionally, since these classifiers are meant to be used in real-time for ultra-wideband localization, both training and inference times are relevant and we use them in our comparisons.

5.2 Hardware

Unless otherwise stated, we used the hardware shown in Table 5.1 for all experimentation and evaluation. For quick experiments, we used a laptop with a 2.9 GHz Intel i7 quad-core CPU and AMD Radeon Pro 560 GPU, with PlaidML GPU accelerated Keras.

Attribute	Value
CPU	Intel® Xeon® E5-2695 v3
CPU Architecture	x86_64
CPU Base Clock Speed	2.3000 GHz
CPU Cores	14 (28 with hyper-threading)
Memory	32 GB
GPU	Nvidia GeForce GTX 980
GPU Memory	4 GB GDDR5
OS	CentOS Linux 7.5.1804 (Core)
Python Version	3.5.5
Keras Version	2.2.4
Tensorflow Version	1.5.0 (tensorflow-gpu)
scikit-learn Version	0.20.0

Table 5.1: Experimental Hardware Configuration

Figure 5.1: LOS Range (scaled, normalized)

5.3 Experimental Results

We trained and evaluated the machine learning model on a total of 71,961 samples collected as described in chapter 4, with 35,976 NLOS and 35,985 LOS samples. An example of a LOS range is shown in Figure 5.1, and a NLOS range in Figure 5.2. This section presents the results we obtained.

5.3.1 SVM (Baseline)

Table 5.2 shows our results for the SVM classifier, which averaged to a 99.97% ($\pm 9.4250 \times 10^{-3}$) cross-validation accuracy, and an average f1 score of 0.9997 ($\pm 9.4290 \times 10^{-5}$). On our hardware, it took 164.0551s to train the model. For validation, it took 29.0916s to make a prediction for 14393 samples, so it took an average of 2.0212 ms per sample to make each prediction. The trained model contained 1108 coefficients.

In order to determine if the nonlinear SVM had high bias or high variance,

Figure 5.2: NLOS Range (scaled, normalized)

Split	Validation Accuracy	f1 score
1	0.99965261	0.99965251
2	0.99951362	0.99951352
3	0.99972207	0.99972211
4	0.99979155	0.99979157
5	0.99972207	0.99972203

Table 5.2: SVM K-Fold Cross-Validation Results

we plotted a learning curve. A learning curve is constructed by training the classifier multiple times on different sized subsets of the data set, and can be used to determine if a model has high bias or variance.

Figure 5.3 shows the learning curve for the SVM classifier with the hyper-parameters optimized. The plot shows a good fit without high bias or variance. The cross validation accuracy increases with the number of training examples, and levels off as it starts to generalize well.

Figure 5.3: Support Vector Machine Learning Curve

5.3.2 Multilayer Perceptron

The simple multilayer-perceptron neural network performed well with an average of 99.9792% (± 0.0100) cross validation accuracy, 0.0039 ($\pm 1.4000 \times 10^{-3}$) loss, and 0.5002 ($\pm 1 \times 10^{-4}$) f1 score. Figure 5.4 and Figure 5.5 show the validation accuracy and loss over a 5-fold cross validation. Figure 5.6 shows the training vs validation accuracy for the first k-fold split, which shows that the MLP did not initially appear to have an overfitting problem. We provide the complete data over all iterations in Appendix C.

Figure 5.4: MLP Neural Network K-Fold Cross Validation Accuracy

Figure 5.5: MLP Neural Network K-Fold Cross Validation Loss

Figure 5.6: MLP Neural Network Training vs Validation Accuracy

The trained model contained 6,307,243 parameters. Training the model took 2183.9487s, and predicting 14,393 samples took 0.9721s, which is an average of 0.0675 ms per sample.

5.3.3 Convolutional Neural Network

The convolutional neural network performed slightly worse but still quite well with an average of 99.9375% (+/- 0.0600) cross validation accuracy, 0.007987 (+/- 8.7000×10^{-3}) loss, and 0.499970 (+/- 3×10^{-4}) f1 score. The validation and loss over a 5-fold cross validation are shown in Figure 5.7 and Figure 5.8.

Similar to the MLP, Figure 5.9 shows that the training and validation accuracy converge to a high accuracy for the first k-fold split. This indicates to us that there is no apparent overfitting problem. The full cross validation results are shown in Appendix C. The trained model contained 4,233 parameters.

Figure 5.7: Convolutional Neural Network K-Fold Cross Validation Accuracy

Figure 5.8: Convolutional Neural Network K-Fold Cross Validation Loss

Using our hardware, training the model took 517.7523s. It took 1.1827s to predict 14,393 samples, so the average prediction time was 0.0822ms per sample.

Figure 5.9: Convolutional Neural Network Training vs Validation Accuracy

5.3.4 LSTM

The long short term memory RNN produced an average cross validation accuracy of 94.3500% (+/- 10.9400), 0.1124 (+/- 0.2108) loss, and 0.4716 (+/- 0.0557) f1 score. Figure 5.10 and Figure 5.11 show the validation accuracy and loss over 5-fold cross validation. It performed well, but it is not as consistent as the MLP and CNN as shown by Split 5. Figure 5.12 shows that the training and validation accuracy converge to a lower accuracy than the MLP and CNN in the first k-fold split, which is an indicator of underfitting. The full cross validation data is available in Appendix C. The trained model contained 273,473 parameters.

Training the model took 4115.6895s on our hardware. Predicting 14,393 samples for validation took 29.1530s, brings us to an average of 2.0255 ms per prediction.

Figure 5.10: LSTM Neural Network K-Fold Cross Validation Accuracy

Figure 5.11: LSTM Neural Network K-Fold Cross Validation Loss

Figure 5.12: LSTM Neural Network Training vs Validation Accuracy

5.4 Testing Results

In this section we will describe the results of evaluating our existing trained models on new data. As described in subsubsection 4.2.2.2, we collected this dataset for the purpose of validating the generalization performance of our deep learning models. A total of 2,000 samples were evaluated on each of our pre-trained deep neural network models.

Model	Accuracy (%)	f1 score
MLP	49.8244	0.0000
CONV	98.9463	0.5077
LSTM	49.3728	0.0010

Table 5.3: Performance of Deep Learning Models on New Data

As shown in Table 5.3, not all of the models generalized well. However, with a 98.9463% accuracy, the convolutional neural network met our success criteria with the new data that was not used for training or validation.

5.5 Discussion

Model	Avg. K-Fold Val. Acc. (%)	Avg. K-fold f1 score	Test Acc. (%)	Train Time (s)	Sample Infer- ence Time (ms)	# of pa- rameters
SVM	99.9680	0.9997	N/A	164.0551	2.0212	1108
MLP	99.9792	0.50012	49.8244	2183.9487	0.0675	6,307,243
CONV	99.9375	0.5000	98.9463	517.7523	0.0822	4,233
LSTM	94.3500	0.4716	49.3728	4115.6895	2.0255	273,473

Table 5.4: Summary of Results

Table 5.4 shows the comparison between each machine learning model we tested using the metrics for our success criteria. We found that both SVMs and neural networks were a good fit for the incorrect leading edge classification

problem, however the two top performing neural networks proved to be faster in inference and comparable in accuracy. Although all three architectures performed with an accuracy of over 95%, the simple multilayer perceptron resulted in the highest k-fold validation accuracy.

The SVM performed well on the data set, but the training time would likely surpass the neural networks as the training set size increases. With a $O(n^3)$ worst case time complexity as more data is added, it's clear that this model will not scale well for production usage.

The convolutional neural network showed both good cross validation accuracy training time, and would be better than the MLP where online training is required. Additionally, it may be useful for memory constrained situations where the large hidden layer of an MLP would not easily fit.

The LSTM network was the most time consuming to train and predict, as it could not be parallelized as effectively and take full advantage of the GPU. It is also apparent that there is room for improvement with this model, and searching more hyper-parameter space could yield better results. Collecting enough data to allow training on sequential data instead of randomly shuffled samples may also be an area to explore, as it may take better advantage of the LSTM architecture.

Since our data is balanced (50% LOS, 50% reflected), the f1 scores is less relevant for our classifier. However, The SVM classifier resulted in the highest f1 score of 0.9997, with the neural network classifiers resulting in comparable f1 scores at ~ 0.5 . The lower f1 score for the neural network classifiers shows that there is less of a balance between precision and recall.

While both the multilayer perceptron and convolutional neural networks met our success criteria with our experimental data, the testing data showed

that only the convolutional neural network generalized well for new data. Since there were a significantly higher number of parameters in the multilayer perceptron and long short-term memory neural networks, it is likely that the trained networks overfit to all of the locations in the experimental data set. However, since the convolutional neural network generalized well and performed as well as the SVM with a faster inference time, we believe it is a good fit for the incorrect leading edge classification problem.

CONCLUSION AND FUTURE WORK

6.1 Contributions

We found in our experimentation that support vector machines, the most popular classifier used for NLOS detection in existing research, fit well for the incorrect leading edge classification problem. With a cross-validation accuracy of 99.96%, and average inference time of 1.7100 ms per sample, our SVM with RBF kernel works well on our small dataset.

While the SVM may have trouble scaling as more training data is collected, deep neural networks have been proven to handle this well. In this work we compared the performance of several neural network models as applied to the misclassified non-line-of-sight problem in ultra-wideband localization, including two types of architectures that to the authors knowledge have not previously been tested in related work. We found that neural networks are a good application for this problem, and one of our models was able to reliably classify ranges as NLOS that the Time Domain ultra-wideband transceiver's onboard NLOS detection misclassified. The convolutional neural network yielded a k-fold cross validation accuracy of 99.9375% and a testing validation accuracy of 98.9463%, which shows promise to improve multipath rejection of UWB localization systems. With a prediction time of 0.0822 ms per sample, our model could be easily deployed for real time scalable predictions. Additionally, this architecture could be ported to FPGA hardware already present on the UWB radios, making it a viable candidate for hardware implementation.

We also found that for online learning where training time is important,

the SVM comes out ahead for this data set with the best training time of 164.0551 s. Our best neural net, the CNN, lags behind with a training time of 517.5752 s. However, it is possible that training time could be improved with hardware optimization.

6.2 Future Work

For future work, more data collection including many more edge cases would be valuable to develop an even more generalized classifier. Additionally, it would be interesting to explore more types of NLOS conditions, such as partially obstructed LOS and fully obstructed direct path where the ranges pass through a solid object. With these additional NLOS states, the complexity of the neural network architectures may need to be increased.

Another interesting direction would be to explore porting these architectures to regressors instead of classifiers to try and estimate the NLOS error. This has been explored in other research such as [33], although to our knowledge other work does not use the direct waveform samples as input.

Finally, in order to apply our best model to production UWB localization applications, it would be useful to port the model to the Time Domain P440 transceiver's onboard FPGA. FPGAs have been shown to be comparable to GPUs for neural network training and inference, with OpenCL being supported on many modern FPGAs allowing the use of high level ML frameworks such as PyTorch [62]. The classifier implemented in FPGA hardware could be used to tag incoming ranges in real time, which would be used upstream in the localization algorithm to mitigate multipath interference.

Bibliography

- [1] Time Domain, *P440 data sheet / user guide*, May 2017.
- [2] R. Ying, T. Jiang, and Z. Xing, “Classification of transmission environment in UWB communication using a support vector machine,” in *Globecom Workshops (GC Wkshps), 2012 IEEE*, IEEE, 2012, pp. 1389–1393.
- [3] Federal Communications Commission, “First report and order 02–48,” Washington, DC, Apr. 22, 2002.
- [4] S. Gezici, Z. Tian, G. B. Giannakis, H. Kobayashi, A. F. Molisch, H. V. Poor, and Z. Sahinoglu, “Localization via ultra-wideband radios: A look at positioning aspects for future sensor networks,” *IEEE Signal Processing Magazine*, vol. 22, no. 4, pp. 70–84, Jul. 2005, ISSN: 1053-5888. DOI: 10.1109/MSP.2005.1458289.
- [5] A. L. Samuel, “Some studies in machine learning using the game of checkers,” *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, Jul. 1959, ISSN: 0018-8646. DOI: 10.1147/rd.33.0210.
- [6] K. Pasupa and W. Sunhem, “A comparison between shallow and deep architecture classifiers on small dataset,” in *2016 8th International Conference on Information Technology and Electrical Engineering (ICITEE)*, Oct. 2016, pp. 1–6. DOI: 10.1109/ICITEED.2016.7863293.
- [7] Google, Inc. (2018). A history of machine learning, (visited on 11/21/2018).
- [8] BBC. (2018). The history of machine learning, BBC Academy, [Online]. Available: <http://www.bbc.co.uk/timelines/zypd97h> (visited on 11/21/2018).
- [9] I. Salian. (Aug. 2, 2018). NVIDIA blog: Supervised vs. unsupervised learning, The Official NVIDIA Blog, [Online]. Available: <https://blogs.nvidia.com/blog/2018/08/02/supervised-unsupervised-learning/> (visited on 12/08/2018).
- [10] scikit-learn developers. (2018). 1.4. support vector machines — scikit-learn 0.20.0 documentation, [Online]. Available: <https://scikit-learn.org/stable/modules/svm.html#complexity> (visited on 11/19/2018).
- [11] V. Vapnik and A. Lerner, “Pattern recognition using generalized portrait method,” *Automation and Remote Control*, vol. 24, pp. 774–780, 1963.

- [12] B. E. Boser, I. M. Guyon, and V. N. Vapnik, “A training algorithm for optimal margin classifiers,” in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, ser. COLT '92, New York, NY, USA: ACM, 1992, pp. 144–152, ISBN: 978-0-89791-497-0. DOI: 10.1145/130385.130401. [Online]. Available: <http://doi.acm.org/10.1145/130385.130401> (visited on 11/21/2018).
- [13] Y.-W. Chang, C.-J. Hsieh, K.-W. Chang, M. Ringgaard, and C.-J. Lin, “Training and testing low-degree polynomial data mappings via linear SVM,” *Journal of Machine Learning Research*, vol. 11, pp. 1471–1490, Apr 2010, ISSN: ISSN 1533-7928. [Online]. Available: <http://jmlr.org/papers/v11/chang10a.html> (visited on 11/19/2018).
- [14] J. J. Hopfield, “Artificial neural networks,” *IEEE Circuits and Devices Magazine*, vol. 4, no. 5, pp. 3–10, Sep. 1988, ISSN: 8755-3996. DOI: 10.1109/101.8118.
- [15] G. E. Hinton, S. Osindero, and Y.-W. Teh, “A fast learning algorithm for deep belief nets,” *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, Jul. 2006, ISSN: 0899-7667. DOI: 10.1162/neco.2006.18.7.1527. [Online]. Available: <http://dx.doi.org/10.1162/neco.2006.18.7.1527> (visited on 11/21/2018).
- [16] J. Brownlee. (May 16, 2016). Crash course on multi-layer perceptron neural networks, Machine Learning Mastery, [Online]. Available: <https://machinelearningmastery.com/neural-networks-crash-course/> (visited on 11/21/2018).
- [17] F. Chollet, *Deep Learning with Python*, 1st. Greenwich, CT, USA: Manning Publications Co., 2017, ISBN: 9781617294433.
- [18] Andrej Karpathy. (Oct. 17, 2018). CS231n convolutional neural networks for visual recognition, [Online]. Available: <http://cs231n.github.io/convolutional-networks/> (visited on 11/21/2018).
- [19] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, Mar. 1994, ISSN: 1045-9227. DOI: 10.1109/72.279181.
- [20] R. Pascanu, T. Mikolov, and Y. Bengio, “Understanding the exploding gradient problem,” *CoRR*, vol. abs/1211.5063, 2012. [Online]. Available: <http://arxiv.org/abs/1211.5063>.
- [21] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1, 1997, ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. [Online]. Available: <https://doi.org/10.1162/neco.1997.9.8.1735> (visited on 11/20/2018).

- [22] V. Pham. (Apr. 18, 2016). Bayesian optimization for hyperparameter tuning, Arimo, [Online]. Available: <https://arimo.com/data-science/2016/bayesian-optimization-hyperparameter-tuning/> (visited on 12/10/2018).
- [23] C.-w. Hsu, C.-c. Chang, and C.-j. Lin, *A practical guide to support vector classification*. 2010.
- [24] Yurii Shevchuk. (Dec. 17, 2016). Hyperparameter optimization for neural networks — NeuPy, Neural Networks in Python, [Online]. Available: http://neupy.com/2016/12/17/hyperparameter_optimization_for_neural_networks.html#disadvantages-of-tpe (visited on 11/12/2018).
- [25] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization,” *Journal of Machine Learning Research*, vol. 13, pp. 281–305, Feb 2012, ISSN: ISSN 1533-7928. [Online]. Available: <http://www.jmlr.org/papers/v13/bergstra12a.html> (visited on 11/12/2018).
- [26] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, “Algorithms for hyperparameter optimization,” in *Proceedings of the 24th International Conference on Neural Information Processing Systems*, ser. NIPS’11, USA: Curran Associates Inc., 2011, pp. 2546–2554, ISBN: 978-1-61839-599-3. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2986459.2986743> (visited on 12/08/2018).
- [27] W. Koehrsen. (Jun. 24, 2018). A conceptual explanation of bayesian hyperparameter optimization for machine learning, Towards Data Science, [Online]. Available: <https://towardsdatascience.com/a-conceptual-explanation-of-bayesian-model-based-hyperparameter-optimization-for-machine-learning-b8172278050f> (visited on 11/12/2018).
- [28] Jason Brownlee. (May 23, 2016). Evaluate the performance of machine learning algorithms in python using resampling, Machine Learning Mastery, [Online]. Available: <https://machinelearningmastery.com/evaluate-performance-machine-learning-algorithms-python-using-resampling/> (visited on 11/12/2018).
- [29] Jeff Schneider. (Feb. 7, 1997). Cross validation, [Online]. Available: <https://www.cs.cmu.edu/~schneide/tut5/node42.html> (visited on 11/11/2018).
- [30] J. Khodjaev, Y. Park, and A. Saeed Malik, “Survey of NLOS identification and error mitigation problems in UWB-based positioning algorithms for dense environments,” *annals of telecommunications - annales des télécommunications*, vol. 65, no. 5, pp. 301–311, Jun. 1, 2010, ISSN: 1958-9395. DOI: 10.1007/s12243-009-0124-z. [Online]. Available: <https://doi.org/10.1007/s12243-009-0124-z> (visited on 11/22/2018).

- [31] B. Chitambira, S. Armour, S. Wales, and M. Beach, “NLOS identification and mitigation for geolocation using least-squares support vector machines,” in *2017 IEEE Wireless Communications and Networking Conference (WCNC)*, Mar. 2017, pp. 1–6. DOI: 10.1109/WCNC.2017.7925566.
- [32] S. Marano, W. M. Gifford, H. Wymeersch, and M. Z. Win, “NLOS identification and mitigation for localization based on UWB experimental data,” *IEEE Journal on Selected Areas in Communications*, vol. 28, no. 7, 2010.
- [33] H. Wymeersch, S. Marano, W. M. Gifford, and M. Z. Win, “A machine learning approach to ranging error mitigation for UWB localization,” *IEEE Transactions on Communications*, vol. 60, no. 6, pp. 1719–1728, 2012.
- [34] W. Li, T. Zhang, and Q. Zhang, “Experimental researches on an UWB NLOS identification method based on machine learning,” in *Communication Technology (ICCT), 2013 15th IEEE International Conference on*, IEEE, 2013, pp. 473–477.
- [35] M. Tabaa, C. Diou, M. El Aroussi, B. Chouri, and A. Dandache, “LOS and NLOS identification based on UWB stable distribution,” in *Microelectronics (ICM), 2013 25th International Conference on*, IEEE, 2013, pp. 1–4.
- [36] S. Tian, L. Zhao, and G. Li, “A support vector data description approach to NLOS identification in UWB positioning,” *Mathematical Problems in Engineering*, vol. 2014, 2014.
- [37] Q. Zhang, D. Zhao, S. Zuo, T. Zhang, and D. Ma, “A low complexity NLOS error mitigation method in UWB localization,” in *2015 IEEE/CIC International Conference on Communications in China (ICCC)*, Nov. 2015, pp. 1–5. DOI: 10.1109/ICCChina.2015.7448714.
- [38] T. V. Nguyen, Y. Jeong, and H. Shin, “Relevance vector machine for UWB localization,” in *2014 IEEE Wireless Communications and Networking Conference (WCNC)*, Apr. 2014, pp. 2150–2155. DOI: 10.1109/WCNC.2014.6952642.
- [39] M. Tipping, “Relevance vector machine,” U.S. Patent 6633857B1, Oct. 14, 2003. [Online]. Available: <https://patents.google.com/patent/US6633857/en> (visited on 11/22/2018).
- [40] X. Yang, F. Zhao, and T. Chen, “NLOS identification for UWB localization based on import vector machine,” *AEU - International Journal of Electronics and Communications*, vol. 87, pp. 128–133, Apr. 1, 2018, ISSN: 1434-8411. DOI: 10.1016/j.aeue.2018.02.003. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S1434841117322379> (visited on 03/30/2018).

- [41] M. Heidari, N. A. Alsindi, and K. Pahlavan, “UDP identification and error mitigation in toa-based indoor localization systems using neural network architecture,” *IEEE Transactions on Wireless Communications*, vol. 8, no. 7, pp. 3597–3607, Jul. 2009, ISSN: 1536-1276. DOI: 10.1109/TWC.2009.080415.
- [42] J. Tiemann, J. Pillmann, and C. Wietfeld, “Ultra-wideband antenna-induced error prediction using deep learning on channel response data,” in *2017 IEEE 85th Vehicular Technology Conference (VTC Spring)*, Jun. 2017, pp. 1–5. DOI: 10.1109/VTCSpring.2017.8108571.
- [43] K. Bregar, A. Hrovat, and M. Mohorčič, “NLOS channel detection with multilayer perceptron in low-rate personal area networks for indoor localization accuracy improvement,” p. 11, May 2016.
- [44] M. Tabaa, C. Diou, R. Saadane, and A. Dandache, “Los/nlos identification based on stable distribution feature extraction and svm classifier for UWB on-body communications,” *Journal of Ubiquitous Systems and Pervasive Networks*, vol. 6, pp. 27–32, 2015.
- [45] X. Cai, X. Li, R. Yuan, and Y. Hei, “Identification and mitigation of NLOS based on channel state information for indoor WiFi localization,” in *Wireless Communications & Signal Processing (WCSP), 2015 International Conference on*, IEEE, 2015, pp. 1–5.
- [46] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [47] C.-C. Chang and C.-J. Lin, “LIBSVM: A library for support vector machines,” *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 3, 27:1–27:27, 2011.
- [48] scikit-learn developers. (2018). RBF SVM parameters — scikit-learn 0.20.0 documentation, [Online]. Available: https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html (visited on 11/19/2018).
- [49] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-I. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snellman, J.

- Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, “In-datacenter performance analysis of a tensor processing unit,” in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17, New York, NY, USA: ACM, 2017, pp. 1–12, ISBN: 978-1-4503-4892-8. DOI: 10.1145/3079856.3080246. [Online]. Available: <http://doi.acm.org/10.1145/3079856.3080246> (visited on 11/19/2018).
- [50] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural Networks*, vol. 2, no. 5, pp. 359–366, Jan. 1, 1989, ISSN: 0893-6080. DOI: 10.1016/0893-6080(89)90020-8. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/0893608089900208> (visited on 11/19/2018).
- [51] keras-team. (Nov. 20, 2018). *Deep learning for humans. contribute to keras-team/keras development by creating an account on GitHub*. original-date: 2015-03-28T00:35:42Z, [Online]. Available: <https://github.com/keras-team/keras> (visited on 11/20/2018).
- [52] M. Pumperla. (Nov. 19, 2018). *Keras + hyperopt: A very simple wrapper for convenient hyperparameter optimization: Maxpumperla/hyperas*. original-date: 2016-02-19T14:45:10Z, [Online]. Available: <https://github.com/maxpumperla/hyperas> (visited on 11/20/2018).
- [53] hyperopt. (Nov. 20, 2018). *Distributed asynchronous hyperparameter optimization in python: Hyperopt/hyperopt*. original-date: 2011-09-06T22:24:59Z, [Online]. Available: <https://github.com/hyperopt/hyperopt> (visited on 11/20/2018).
- [54] J. Brownlee. (Sep. 20, 2018). How to develop 1d convolutional neural network models for human activity recognition, Machine Learning Mastery, [Online]. Available: <https://machinelearningmastery.com/cnn-models-for-human-activity-recognition-time-series-classification/> (visited on 11/20/2018).
- [55] Y. LeCun, Y. Bengio, *et al.*, “Convolutional networks for images, speech, and time series,” *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.
- [56] C. Metz, “With QuickType, apple wants to do more than guess your next text. it wants to give you an AI,” *Wired*, Jun. 14, 2016, ISSN: 1059-1028. [Online]. Available: <https://www.wired.com/2016/06/apple-bringing-ai-revolution-iphone/> (visited on 11/20/2018).
- [57] A. Graves, A. Mohamed, and G. Hinton, “Speech recognition with deep recurrent neural networks,” in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2013, pp. 6645–6649. DOI: 10.1109/ICASSP.2013.6638947.

- [58] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, “LSTM: A search space odyssey,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, Oct. 2017, ISSN: 2162-237X, 2162-2388. DOI: 10.1109/TNNLS.2016.2582924. arXiv: 1503.04069. [Online]. Available: <http://arxiv.org/abs/1503.04069> (visited on 11/20/2018).
- [59] Time Domain, *Broadspec™ uwb antenna*, Jun. 2017.
- [60] —, *Rangenet user guide pulson® 400 series*, Sep. 2017.
- [61] N. Chinchor, “MUC-4 evaluation metrics,” in *Proceedings of the 4th Conference on Message Understanding*, ser. MUC4 ’92, Stroudsburg, PA, USA: Association for Computational Linguistics, 1992, pp. 22–29, ISBN: 978-1-55860-273-1. DOI: 10.3115/1072064.1072067. [Online]. Available: <https://doi.org/10.3115/1072064.1072067> (visited on 11/20/2018).
- [62] G. Lacey, G. W. Taylor, and S. Areibi, “Deep learning on FPGAs: Past, present, and future,” *arXiv:1602.04283 [cs, stat]*, Feb. 12, 2016. arXiv: 1602.04283. [Online]. Available: <http://arxiv.org/abs/1602.04283> (visited on 11/21/2018).
- [63] Y. Quan, L. Lau, A. Wen, F. Jing, Q. Nie, and S.-Y. Cho, “Analysis and machine-learning based detection of outlier measurements of ultra-WideBand in an obstructed environment,”
- [64] J. Wang, Q. Gao, Y. Yu, H. Wang, and M. Jin, “Toward robust indoor localization based on bayesian filter using chirp-spread-spectrum ranging,” *IEEE Transactions on Industrial Electronics*, vol. 59, no. 3, pp. 1622–1629, 2012.
- [65] T. Van Nguyen, Y. Jeong, H. Shin, and M. Z. Win, “Machine learning for wideband localization,” *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 7, pp. 1357–1380, 2015.
- [66] T. Van Nguyen, Y. Jeong, and H. Shin, “Relevance vector machine for UWB localization,” in *Wireless Communications and Networking Conference (WCNC), 2014 IEEE*, IEEE, 2014, pp. 2150–2155.
- [67] Z. Yin, K. Cui, Z. Wu, and L. Yin, “Entropy-based TOA estimation and SVM-based ranging error mitigation in UWB ranging systems,” *Sensors*, vol. 15, no. 5, pp. 11 701–11 724, 2015.
- [68] X. Luo, X. Chang, and H. Liu, “A taylor based localization algorithm for wireless sensor network using extreme learning machine,” *IE-ICE TRANSACTIONS on Information and Systems*, vol. 97, no. 10, pp. 2652–2659, 2014.

- [69] W. Zhang, L. Wang, Z. Qin, X. Zheng, L. Sun, N. Jin, and L. Shu, "INBS: An improved naive bayes simple learning approach for accurate indoor localization," in *Communications (ICC), 2014 IEEE International Conference on*, IEEE, 2014, pp. 148–153.
- [70] T. Wehs, A. C. Maheri, and C. Koch, "Classification of ultra-wideband (UWB) pulses distorted by penetration effects,"
- [71] K. Bregar, A. Hrovat, and M. Mohorčič, "NLOS channel detection with multilayer perceptron in low-rate personal area networks for indoor localization accuracy improvement,"
- [72] X. Li, X. Cai, Y. Hei, and R. Yuan, "NLOS identification and mitigation based on channel state information for indoor WiFi localisation," *IET Communications*, vol. 11, no. 4, pp. 531–537, 2017.
- [73] A. Ledergerber and R. D'Andrea, "Ultra-wideband range measurement model with gaussian processes," in *Control Technology and Applications (CCTA), 2017 IEEE Conference on*, IEEE, 2017, pp. 1929–1934.
- [74] Z.-M. Miao, L.-W. Zhao, W.-W. Yuan, and F.-L. Jin, "Application of one-class classification in NLOS identification of UWB positioning," in *Information System and Artificial Intelligence (ISAI), 2016 International Conference on*, IEEE, 2016, pp. 318–322.
- [75] J. Wu, T. Zhao, S. Li, and C.-M. Own, "Belief interval of dempster-shafer theory for line-of-sight identification in indoor positioning applications," *Sensors*, vol. 17, no. 6, p. 1242, 2017.
- [76] H. Soleimani, S. Tomasin, T. Alizadeh, and M. Shojafar, "Cluster-head based feedback for simplified time reversal prefiltering in ultra-wideband systems," *Physical Communication*, vol. 25, pp. 100–109, 2017.
- [77] Time Domain, *Rangenet application programming interface (api) specification*, Sep. 2017.

APPENDICES

Appendix A

CODE LISTING

```
1  #!/usr/bin/env python3
2  import numpy as np
3  import csv
4  import os
5  import pandas as pd
6  import datetime
7  import logging
8  import pickle
9  import time
10 from sys import platform
11 import matplotlib.pyplot as plt
12
13 # if platform == "darwin":
14 #     # use plaid-ml on Macs to take advantage of AMD GPUs
15 #     import plaidml.keras
16 #     plaidml.keras.install_backend()
17 # else:
18 from tensorflow import set_random_seed
19 from keras.models import Sequential, load_model
20 from keras.layers import Dense, Dropout, Flatten, Activation
21 from keras.layers import LeakyReLU, Conv1D, MaxPool1D, CuDNNLSTM
22 from keras.callbacks import CSVLogger, TensorBoard
23 from keras.constraints import maxnorm
24 from keras import optimizers
25 from keras import regularizers
26 from keras import backend as K
27
28
29 from hyperas import optim
30 from hyperas.distributions import choice, uniform
31 from hyperas.utils import eval_hypermopt_space
32
33 from hyperopt import Trials, STATUS_OK, tpe
34 from sklearn.model_selection import learning_curve, train_test_split, \
35     StratifiedKFold, StratifiedShuffleSplit, GridSearchCV, \
36     RandomizedSearchCV, cross_validate
37 from sklearn.svm import SVC
38 from sklearn.metrics import classification_report, confusion_matrix, f1_score, \
39     precision_score, recall_score
40 from sklearn.preprocessing import StandardScaler
41
42 logging.basicConfig(level=logging.DEBUG)
43 log = logging
44 seed = 812
45
```

```

46
47 class FullScanInfo:
48     def __init__(self, time, msg_id, src_id, emb_time, noise, v_peak,
49                 leading_edge_offset, lockspot_offset, scan_start_ps,
50                 scan_stop_ps, scan_step_bins, filtering, ant_id, num_samples,
51                 scan_data):
52         """
53         Creates a Time Domain FULL_SCAN_INFO object, as described in the
54         RangeNet API specification.
55         """
56         self.time = time
57         self.msg_id = msg_id
58         self.src_id = src_id
59         self.emb_time = emb_time
60         self.noise = noise
61         self.v_peak = v_peak
62         self.leading_edge_offset = leading_edge_offset
63         self.lockspot_offset = lockspot_offset
64         self.scan_start_ps = scan_start_ps
65         self.scan_stop_ps = scan_stop_ps
66         self.scan_step_bins = scan_step_bins
67         self.filtering = filtering
68         self.ant_id = ant_id
69         self.num_samples = num_samples
70         self.scan_data = scan_data
71
72     def get_feature_array(self):
73         """
74         Gets an array of time independent features as collected from the
75         Time Domain radios.
76         :return: np.array
77         """
78         return np.array([self.noise, self.v_peak, self.leading_edge_offset,
79                         self.lockspot_offset])
80
81
82 class RangeInfo:
83     def __init__(self, time, msg_id, resp_id, status, req_ant_mode,
84                 resp_ant_mode, stopwatch_time, precision_range_mm,
85                 course_range_mm, filtered_range_mm, precision_range_err_mm,
86                 course_range_err_mm, filtered_range_err_mm,
87                 filtered_range_vel_mm_per_sec,
88                 filtered_range_vel_mm_per_sec_err, range_type,
89                 req_led_flags, resp_led_flags, noise, v_peak,
90                 course_tof_in_bins, embedded_timestamp, reserved):
91         """
92         Creates a Time Domain RANGE_INFO object, as described in the
93         RangeNet API specification.
94         """
95         self.time = time
96         self.msg_id = msg_id
97         self.resp_id = resp_id
98         self.status = status
99         self.req_ant_mode = req_ant_mode
100        self.resp_ant_mode = resp_ant_mode

```

```

101         self.stopwatch_time = stopwatch_time
102         self.precision_range_mm = precision_range_mm
103         self.course_range_mm = course_range_mm
104         self.filtered_range_mm = filtered_range_mm
105         self.precision_range_err_mm = precision_range_err_mm
106         self.course_range_err_mm = course_range_err_mm
107         self.filtered_range_err_mm = filtered_range_err_mm
108         self.filtered_range_vel_mm_per_sec = filtered_range_vel_mm_per_sec
109         self.filtered_range_vel_mm_per_sec_err = filtered_range_vel_mm_per_sec_err
110         self.range_type = range_type
111         self.req_led_flags = req_led_flags
112         self.resp_led_flags = resp_led_flags
113         self.noise = noise
114         self.v_peak = v_peak
115         self.course_tof_in_bins = course_tof_in_bins
116         self.embedded_timestamp = embedded_timestamp
117         self.reserved = reserved
118
119
120     def ranges_to_features(ranges_los, ranges_nlos, channel_stats=False,
121                           samples=True):
122         """
123         Takes two arrays of raw waveform samples, LOS samples, and NLOS samples.
124         Returns an array of both sets of samples, and an array of the labels
125         for that array, where 1 is a LOS range, and 0 is a NLOS range.s
126         """
127         if samples:
128             # extract waveforms from inputs
129             ranges_los = np.array(
130                 list(map(lambda fr: fr.scan_data, ranges_los)))
131             labels_los = np.ones(len(ranges_los))
132             rngs_rflctd = np.array(
133                 list(map(lambda fr: fr.scan_data, ranges_nlos)))
134             labels_reflected = np.zeros(len(ranges_nlos))
135
136             inputs = np.concatenate(
137                 (ranges_los.astype(np.float32), rngs_rflctd.astype(np.float32)))
138             labels = np.concatenate(
139                 (
140                     labels_los.astype(np.float32), labels_reflected.astype(np.float32)))
141         else:
142             # extract channel stats from inputs
143             ## FIX need to scale each feature independently
144             ranges_los = np.array(
145                 list(map(lambda fr: fr.get_feature_array(), ranges_los)))
146             labels_los = np.ones(len(ranges_los))
147             rngs_rflctd = np.array(
148                 list(map(lambda fr: fr.get_feature_array(), ranges_nlos)))
149             labels_reflected = np.zeros(len(rngs_rflctd))
150
151             inputs = np.concatenate(
152                 (ranges_los.astype(np.float32), rngs_rflctd.astype(np.float32)))
153             labels = np.concatenate(
154                 (
155                     labels_los.astype(np.float32), labels_reflected.astype(np.float32)))

```

```

156
157     return inputs, labels
158
159
160 def scale_inputs(inputs, std=None, mean=None):
161     """
162     Scales and normalizes UWB waveform samples. For each sample:
163         - subtract the mean of all samples
164         - divide by the std of all samples
165     :param inputs: array of waveform samples
166     :param std: (optional) instead of calculating the std of the inputs,
167                 use this std
168     :param mean: instead of calculating the mean of the inputs, use this mean
169     :return: a tuple of: the scaled & normalized inputs, the std of the inputs,
170                and the mean of the inputs
171     """
172     # scale inputs
173     inputs_flat = inputs.flatten()
174
175     if mean:
176         inputs_mean = mean
177     else:
178         inputs_mean = inputs_flat.mean()
179     if std:
180         inputs_std = std
181     else:
182         inputs_std = inputs_flat.std()
183
184     inputs = np.subtract(inputs, inputs_mean)
185     inputs = np.divide(inputs, inputs_std)
186
187     return inputs, inputs_std, inputs_mean
188
189
190 def get_keras_loggers(model_name):
191     """
192     Builds a CSV and Tensorflow logger for a Keras model.
193
194     :param model_name: name of the model to use for the logging file names
195     :return: an array of Keras loggers
196     """
197     loggers = []
198
199     if not os.path.isdir("logs/logs_{}".format(model_name)):
200         os.mkdir("logs/logs_{}".format(model_name))
201
202     # always log to csv
203     csv_log = CSVLogger(
204         "logs/logs_{}/{}.csv".format(model_name,
205                                     datetime.datetime.now().isoformat())
206     )
207     loggers.append(csv_log)
208
209     if platform != "darwin":
210         tb = TensorBoard(log_dir="logs/logs_{}/{}".format(model_name,
211                                                             datetime.datetime.now().isoformat()),

```

```

211             histogram_freq=0, write_grads=False)
212         loggers.append(tb)
213
214     return loggers
215
216
217 def f1_score(y_true, y_pred):
218     """
219     F1 score custom metrics function for Keras.
220     From https://stackoverflow.com/a/45305384/3047919
221     :param y_true:
222     :param y_pred:
223     :return:
224     """
225     def recall(y_true, y_pred):
226         """Recall metric.
227
228         Only computes a batch-wise average of recall.
229
230         Computes the recall, a metric for multi-label classification of
231         how many relevant items are selected.
232         """
233         true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
234         possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
235         recall = true_positives / (possible_positives + K.epsilon())
236         return recall
237
238     def precision(y_true, y_pred):
239         """Precision metric.
240
241         Only computes a batch-wise average of precision.
242
243         Computes the precision, a metric for multi-label classification of
244         how many selected items are relevant.
245         """
246         true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
247         predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
248         precision = true_positives / (predicted_positives + K.epsilon())
249         return precision
250     precision = precision(y_true, y_pred)
251     recall = recall(y_true, y_pred)
252     return 2*((precision*recall)/(precision+recall+K.epsilon()))
253
254
255 def build_fit_model_mlp(x_train, y_train, x_val, y_val, save_model=False, hyper_opt=False):
256     """
257     Keras model function for the MLP. Builds and fits the model.
258     Note: function definition needs to be on one line or it messed up hyperas.
259     :param x_train: training input data
260     :param y_train: training input labels
261     :param x_val: validation input data
262     :param y_val: validation input labels
263     :param save_model: if the model should be saved to disk after training
264     :param hyper_opt: if you are calling this function using hyperas
265     :return: the trained Keras model

```

```

266     """
267     samples = x_train[0].size
268
269     # hyper-params!
270     if hyper_opt:
271         dropout = {{uniform(0, 0.5)}}
272         if platform == "darwin": # plaid-ml doesn't support this
273             constr = None
274         else:
275             constr = {{choice([None, maxnorm(3), maxnorm(4), maxnorm(5)])}}
276         # had to hardcode because hyperas is dumb :|
277         neurons_hidden = {{choice(np.arange(1632 // 4, 1632 * 2, dtype=int))}}
278         batch_size = {{choice([32, 64, 128])}}
279         optimizer = {{choice(['adam', 'sgd', 'rmsprop'])}}
280         learning_rate = {{choice(np.logspace(-6, -1, 6))}}
281         epochs = {{choice([25, 50, 100])}}
282         use_regularization = {{choice([True, False])}}
283         reg_lambda = {{choice(np.logspace(-5, -1, 5))}}
284         inner_activation = {{choice([Activation('relu'), LeakyReLU()])}}
285         inner_activation_1 = inner_activation
286         inner_activation_2 = inner_activation
287     else:
288         dropout = 0.3012810523189274
289         constr = None
290         neurons_hidden = 2229
291         batch_size = 32
292         optimizer = 'adam'
293         learning_rate = 0.00001
294         epochs = 100
295         use_regularization = True
296         reg_lambda = 0.00001
297         inner_activation_1 = Activation('relu')
298         inner_activation_2 = Activation('relu')
299
300     # regularizer
301     reg = None
302     if use_regularization:
303         reg = regularizers.l2(reg_lambda)
304
305     # optimizer
306     if optimizer == 'adam':
307         opt = optimizers.adam(lr=learning_rate)
308     elif optimizer == 'sgd':
309         opt = optimizers.sgd(lr=learning_rate)
310     elif optimizer == 'rmsprop':
311         opt = optimizers.rmsprop(lr=learning_rate)
312     else:
313         opt = optimizers.adam(lr=learning_rate)
314
315     model = Sequential()
316     model.add(Dense(samples, input_dim=samples, kernel_constraint=constr,
317                     kernel_regularizer=reg))
318     model.add(inner_activation_1)
319     model.add(Dropout(dropout))
320

```



```

321     model.add(Dense(neurons.hidden, kernel_constraint=constr,
322                     kernel_regularizer=reg))
323     model.add(inner_activation_2)
324     model.add(Dropout(dropout))
325
326     model.add(Dense(1, activation='sigmoid'))
327
328     # Compile model
329     model.compile(loss='binary_crossentropy', optimizer=opt,
330                  metrics=['accuracy', f1_score])
331
332     if not hyper_opt:
333         loggers = get_keras_loggers('mlp')
334     else:
335         loggers = [CSVLogger("logs/logs_{}/{}/.csv".format('mlp',
336                                                             datetime.datetime.now().isoformat()))]
337
338     # fit the model
339     res = model.fit(x_train, y_train, epochs=epochs, batch_size=batch_size,
340                   validation_data=(x_val, y_val),
341                   callbacks=loggers)
342
343     if save_model:
344         model.save('models/model_mlp.h5')
345
346     if hyper_opt:
347         validation_acc = np.amax(res.history['val_acc'])
348         print('Best_validation_acc_of_epoch:', validation_acc)
349         return {'loss': -validation_acc, 'status': STATUS_OK, 'model': model}
350     else:
351         return model
352
353
354 def build_fit_model_conv(x_train, y_train, x_val, y_val, save_model=False, hyper_opt=False):
355     """
356     Keras model function for the CNN. Builds and fits the model.
357     Note: function definition needs to be on one line or it messed up hyperas.
358     :param x_train: training input data
359     :param y_train: training input labels
360     :param x_val: validation input data
361     :param y_val: validation input labels
362     :param save_model: if the model should be saved to disk after training
363     :param hyper_opt: if you are calling this function using hyperas
364     :return: the trained Keras model
365     """
366     samples = x_train.shape[1]
367
368     # 3-d-ify
369     train_shape = x_train.shape + (1,)
370     val_shape = x_val.shape + (1,)
371     x_train_3d = np.reshape(x_train, train_shape)
372     x_val_3d = np.reshape(x_val, val_shape)
373
374     # hyper-params!
375     if hyper_opt:

```

```

376     kernel_size = {{choice([3, 4, 5, 6, 7, 8])}}
377     strides_first = {{choice(list(range(1, 5)))}} # because hyperopt can't even
378     strides_second = {{choice([1, 2, 3, 4])}}
379     strides_pool = {{choice([1, 2, 3, 4, 5, 6, 7, 8])}}
380     neurons_first = {{choice([32, 64, 128])}}
381     neurons_second = {{choice([8, 16, 32])}}
382     batch_size = {{choice([16, 32, 64])}}
383     optimizer = {{choice(['adam', 'sgd', 'rmsprop'])}}
384     learning_rate = {{choice(np.logspace(-6, -1, 6))}}
385     epochs = {{choice([25, 50, 100])}}
386     inner_activation = {{choice([Activation('relu'), LeakyReLU()])}}
387     inner_activation_1 = inner_activation
388     inner_activation_2 = inner_activation
389 else:
390     kernel_size = 7
391     strides_first = 2
392     strides_second = 1
393     strides_pool = 7
394     neurons_first = 64
395     neurons_second = 8
396     batch_size = 32
397     optimizer = 'adam'
398     learning_rate = 0.0001
399     epochs = 50
400     inner_activation_1 = LeakyReLU()
401     inner_activation_2 = LeakyReLU()
402
403 # optimizer
404 if optimizer == 'adam':
405     opt = optimizers.adam(lr=learning_rate)
406 elif optimizer == 'sgd':
407     opt = optimizers.sgd(lr=learning_rate)
408 elif optimizer == 'rmsprop':
409     opt = optimizers.rmsprop(lr=learning_rate)
410 else:
411     opt = optimizers.adam(lr=learning_rate)
412
413 model = Sequential()
414 model.add(Conv1D(neurons_first, kernel_size=kernel_size,
415                 strides=strides_first, input_shape=(samples, 1)))
416 model.add(inner_activation_1)
417 model.add(MaxPool1D(pool_size=2, strides=strides_pool))
418
419 model.add(Conv1D(neurons_second, kernel_size=kernel_size,
420                 strides=strides_second))
421 model.add(inner_activation_2)
422 model.add(MaxPool1D(pool_size=2, strides=strides_pool))
423
424 model.add(Flatten())
425 model.add(Dense(1, activation='sigmoid'))
426
427 # Compile model
428 model.compile(loss='binary_crossentropy', optimizer=opt,
429              metrics=['accuracy', 'f1_score'])
430

```

```

431     # loggers
432     if not hyper_opt:
433         loggers = get_keras_loggers('conv')
434     else:
435         loggers = [CSVLogger("logs/logs_{}/{}/{}.csv".format('conv',
436             datetime.datetime.now().isoformat()))]
437
438     # fit the model
439     res = model.fit(x_train_3d, y_train, epochs=epochs, batch_size=batch_size,
440         validation_data=(x_val_3d, y_val),
441         callbacks=loggers)
442
443     if save_model:
444         model.save('models/model_conv.h5')
445
446     if hyper_opt:
447         validation_acc = np.amax(res.history['val_acc'])
448         print('Best-validation_acc-of-epoch:', validation_acc)
449         return {'loss': -validation_acc, 'status': STATUS_OK, 'model': model}
450     else:
451         return model
452
453
454 def build_fit_model_lstm(x_train, y_train, x_val, y_val, save_model=False, hyper_opt=False):
455     """
456     Keras model function for the LSTM. Builds and fits the model.
457     Note: function definition needs to be on one line or it messed up hyperas.
458     :param x_train: training input data
459     :param y_train: training input labels
460     :param x_val: validation input data
461     :param y_val: validation input labels
462     :param save_model: if the model should be saved to disk after training
463     :param hyper_opt: if you are calling this function using hyperas
464     :return: the trained Keras model
465     """
466     samples = x_train.shape[1]
467
468     # 3-d-ify
469     x_train_shape = x_train.shape + (1,)
470     x_train_3d = np.reshape(x_train, x_train_shape)
471     x_val_shape = x_val.shape + (1,)
472     x_val_3d = np.reshape(x_val, x_val_shape)
473
474     # hyper-params!
475     if hyper_opt:
476         dropout = [{uniform(0, 0.5)}]
477         if platform == "darwin": # plaid-ml doesn't support this
478             constr = None
479         else:
480             constr = [{choice([None, maxnorm(3), maxnorm(4), maxnorm(5)])}]
481         # had to hardcode because hyperas is dumb :)
482         neurons_hidden = [{choice([32, 64, 128])}]
483         batch_size = [{choice([8, 16, 32])}]
484         optimizer = [{choice(['rmsprop'])}]
485         learning_rate = [{choice(np.logspace(-6, -1, 6))}]

```

```

486         epochs = {{choice([10, 15, 20, 40])}}
487         use_regularizer = {{choice([True, False])}}
488         reg_lambda = {{choice(np.logspace(-5, -1, 5))}}
489         inner_activation = {{choice([Activation('relu'), LeakyReLU()])}}
490         lstm_output = {{choice([64, 128, 256])}}
491     else:
492         dropout = 0.1603763880022983
493         constr = maxnorm(5)
494         neurons_hidden = 32
495         batch_size = 32
496         optimizer = 'rmsprop'
497         learning_rate = 0.001
498         epochs = 15
499         use_regularizer = True
500         reg_lambda = 0.0001
501         inner_activation = LeakyReLU()
502         lstm_output = 256
503
504     # regularizer
505     reg = None
506     if use_regularizer:
507         reg = regularizers.l2(reg_lambda)
508
509     # Compile model
510     if optimizer == 'adam':
511         opt = optimizers.adam(lr=learning_rate)
512     elif optimizer == 'sgd':
513         opt = optimizers.sgd(lr=learning_rate)
514     elif optimizer == 'rmsprop':
515         opt = optimizers.rmsprop(lr=learning_rate)
516     else:
517         opt = optimizers.adam(lr=learning_rate)
518
519     model = Sequential()
520     model.add(CuDNNLSTM(lstm_output, input_shape=(samples, 1),
521                        return_sequences=False))
522     model.add(Dropout(dropout))
523     model.add(Dense(neurons_hidden, kernel_regularizer=reg,
524                    kernel_constraint=constr))
525     model.add(inner_activation)
526
527     model.add(Dropout(dropout))
528     model.add(Dense(1, activation='sigmoid'))
529
530     model.compile(loss='binary_crossentropy', optimizer=opt,
531                  metrics=['accuracy', f1_score])
532
533     if not hyper_opt:
534         loggers = get_keras_loggers('lstm')
535     else:
536         loggers = [CSVLogger("logs/logs_{{}}/{}.csv".format('lstm',
537                    datetime.datetime.now().isoformat()))]
538
539     # fit the model
540     res = model.fit(x_train_3d, y_train, epochs=epochs, batch_size=batch_size,

```

```

541         validation_data=(x_val_3d, y_val),
542         callbacks=loggers)
543
544     if save_model:
545         model.save('models/model_lstm.h5')
546
547     if hyper_opt:
548         validation_acc = np.amax(res.history['val_acc'])
549         print('Best_validation_acc_of_epoch:', validation_acc)
550         return {'loss': -validation_acc, 'status': STATUS.OK, 'model': model}
551     else:
552         return model
553
554
555 def train_nn(inputs, labels, kfold=False, save_model=False, hyper_opt=False,
556             hyper_opt_max_evals=10):
557     """
558     Trains and tests a neural network using Keras. Supports k-fold
559     cross-validation and hyperas hyper-paramter optimization.
560     :param inputs: an array of input samples
561     :param labels: labels for the input samples
562     :param kfold: if you would like to perform 5-fold cross-validation
563     :param save_model: if the model should be saved after training
564     :param hyper_opt: if you would like to perform hyperas hyper-paramter
565                      optimization
566     :param hyper_opt_max_evals: the maximum iterations for hyper-param opt
567     :return: the trained Keras model
568     """
569     model = None
570     model_func = build_fit_model_mlp
571     model_name = 'mlp'
572
573     # quick testing
574     # inputs = np.concatenate((inputs[:32], inputs[-32:]))
575     # labels = np.concatenate((labels[:32], labels[-32:]))
576
577     if hyper_opt:
578         trials = Trials()
579         best_run, best_model, space = optim.minimize(model=model_func,
580                                                     data=data_func.hyperas,
581                                                     algo=tpe.suggest,
582                                                     max_evals=hyper_opt_max_evals,
583                                                     trials=trials,
584                                                     eval_space=True,
585                                                     return_space=True)
586         print("Best_performing_model_chosen_hyper-parameters:")
587         print(best_run)
588
589         csv_headers = ['trial', 'loss']
590         csv_headers.extend(sorted(trials.trials[0]['misc']['vals'].keys()))
591         csv_rows = [csv_headers]
592         for t, trial in enumerate(trials):
593             csv_row = [t, trial['result']['loss']]
594             vals = trial.get('misc').get('vals')
595             hyperspace_dick = eval.hyperopt_space(space, vals)

```

```

596         csv_row.extend([hyperspace_dick[key] for key in sorted(hyperspace_dick.keys())])
597     csv_rows.append(csv_row)
598     with open("log_%s_hyperopt.csv" % model_name, 'w') as csv_file:
599         writer = csv.writer(csv_file)
600         writer.writerows(csv_rows)
601     elif kfold:
602         scores = []
603
604         kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=seed)
605         for train_ndxs, val_ndxs in kfold.split(inputs, labels):
606             x_train, x_val = inputs[train_ndxs], inputs[val_ndxs]
607             y_train, y_val = labels[train_ndxs], labels[val_ndxs]
608
609             model = model_func(x_train, y_train, x_val, y_val,
610                               save_model=save_model)
611             # CONV NN & LSTM
612             if model_name == 'conv' or model_name == 'lstm':
613                 score = model.evaluate(np.reshape(x_val, x_val.shape + (1,)), y_val)
614             else:
615                 score = model.evaluate(x_val, y_val)
616             print("\nTEST_result: %s: %.4f %s: %.4f%%\n" % (model.metrics_names[0],
617                                                            score[0],
618                                                            model.metrics_names[1],
619                                                            score[1] * 100))
620             scores.append(score)
621
622         losses = [score[0] for score in scores]
623         accuracies = [score[1] for score in scores]
624         f1_scores = [score[2] for score in scores]
625         print("kfold_avg_acc: %.6f%%(+/-%.4f%%)\n" % (np.mean(accuracies),
626                                                       np.std(accuracies)))
627         print("kfold_avg_loss: %.6f(+/-%.4f%%)\n" % (np.mean(losses),
628                                                       np.std(losses)))
629         print("kfold_avg_f1: %.6f(+/-%.4f%%)\n" % (np.mean(f1_scores),
630                                                       np.std(f1_scores)))
631     else:
632         x_train, x_val, \
633         y_train, y_val = train_test_split(inputs, labels,
634                                           test_size=0.2, random_state=seed)
635
636         fit_start = time.time()
637         model = model_func(x_train, y_train, x_val, y_val,
638                           save_model=save_model)
639         fit_finish = time.time()
640
641         pred_start = time.time()
642         if model_name == 'conv' or model_name == 'lstm':
643             score = model.evaluate(np.reshape(x_val, x_val.shape + (1,)), y_val)
644         else:
645             score = model.evaluate(x_val, y_val)
646         pred_finish = time.time()
647
648         print("\nt_fit_%d_samples: %s\n" % (len(x_train),
649                                             str(fit_finish - fit_start)))
650         print("\nt_predict_%d_samples: %s\n" % (len(x_val),

```

```

651         str(pred_finish - pred_start)))
652
653     print("TEST_%s: _%.6f_%s: _%.6f%%_%.6f\n" % (model.metrics_names[0],
654                                                    score[0],
655                                                    model.metrics_names[1],
656                                                    score[1] * 100,
657                                                    model.metrics_names[2],
658                                                    score[2]))
659
660     return model
661
662
663 def train_svm(inputs, labels, kfold=False, save_model=False, hyper_opt=False,
664              calc_learning_curve=False):
665     """
666     Trains and tests a support vector machine classifier. Supports 5-fold
667     cross-validation and random search hyper-parameter optimization.
668     :param inputs: an array of input data for the SVM
669     :param labels: labels for the input data
670     :param kfold: if you would like to perform hyper-parameter optimization
671     :param save_model: if the model should be saved to disk after training
672     :param hyper_opt: if you would like to perform hyper-parameter optimization
673     :param calc_learning_curve: if you would like to calculate the
674                               learning curve
675     """
676     svm_kernel = 'rbf'
677     svm_cache_size = 500
678
679     best_C = 1000.0
680     best_gamma = 0.001
681
682     # quick testing
683     # inputs = np.concatenate((inputs[:32], inputs[-32:]))
684     # labels = np.concatenate((labels[:32], labels[-32:]))
685
686     if hyper_opt:
687         model = SVC(kernel=svm_kernel, verbose=True, cache_size=svm_cache_size)
688
689         # random search hyper-parameter optimization
690         C_range = np.logspace(-3, 3, 7)
691         gamma_range = np.logspace(-3, 3, 7)
692         param_grid = dict(gamma=list(gamma_range), C=list(C_range))
693         cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=807)
694         # grid_search = GridSearchCV(model, param_grid, n_jobs=-1, iid=False,
695         #                             refit=False, cv=cv, verbose=2,
696         #                             return_train_score=False)
697         rand_search = RandomizedSearchCV(model, param_grid, n_iter=20, cv=cv,
698                                         iid=False, refit=False,
699                                         return_train_score=False,
700                                         n_jobs=-1, pre_dispatch='2*n_jobs',
701                                         verbose=2)
702         rand_search.fit(inputs, labels)
703
704     pd.DataFrame(rand_search.cv_results_).to_csv("svm_rand_search.csv")
705

```

```

706     print("The best parameters are %s with a score of %0.6f"
707           % (rand_search.best_params_, rand_search.best_score_))
708 elif kfold:
709     model = SVC(kernel=svm_kernel, verbose=True, cache_size=svm_cache_size,
710                 C=best_C, gamma=best_gamma)
711     cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=807)
712     scoring = {'acc': 'accuracy',
713               'precision': 'precision',
714               'recall': 'recall',
715               'f1': 'f1'}
716     scores = cross_validate(model, inputs, y=labels, cv=cv,
717                             scoring=scoring, verbose=1,
718                             n_jobs=-1, pre_dispatch='2*n_jobs')
719
720     print("scores: %s\n" % str(scores))
721     print("avg_scores: %s\n" % str({k: np.average(v) for k, v in scores.items()}))
722     print("std_scores: %s\n" % str({k: np.std(v) for k, v in scores.items()}))
723 elif calc_learning_curve:
724     model = SVC(kernel=svm_kernel, verbose=True, cache_size=svm_cache_size,
725                 C=best_C, gamma=best_gamma)
726
727     # kfold learning curve
728     cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=807)
729     train_sizes, train_scores, \
730     val_scores = learning_curve(model, inputs, labels, cv=cv,
731                                 shuffle=True, error_score='raise',
732                                 n_jobs=-1, pre_dispatch='2*n_jobs')
733     print("Train_scores:_" + str(train_scores))
734     print("Val_scores:_" + str(val_scores))
735     print("sizes:_" + str(train_sizes))
736
737     train_scores_avg = np.matrix(np.mean(train_scores, axis=1))
738     val_scores_avg = np.matrix(np.mean(val_scores, axis=1))
739     sizes = np.matrix(train_sizes)
740
741     train_scores_output = np.concatenate((sizes, train_scores_avg)).T
742     val_scores_output = np.concatenate((sizes, val_scores_avg)).T
743     np.savetxt("log_svm_learning_curve_train.csv", train_scores_output, delimiter=",")
744     np.savetxt("log_svm_learning_curve_val.csv", val_scores_output, delimiter=",")
745 else:
746     model = SVC(kernel=svm_kernel, verbose=True, cache_size=svm_cache_size,
747                 C=best_C, gamma=best_gamma)
748
749     x_train, x_test, y_train, y_test = train_test_split(inputs, labels,
750                                                         test_size=0.2)
751     # train
752     fit_start = time.time()
753     model.fit(x_train, y_train)
754     fit_finish = time.time()
755     if save_model:
756         pickle.dump(model, open("models/svm_rbf.obj", "wb"))
757
758     # test
759     pred_start = time.time()
760     y_pred = model.predict(x_test)

```



```

761     pred_finish = time.time()
762
763     print("\nt_fit_%d_samples: %s\n" % (len(x_train),
764                                         str(fit_finish - fit_start)))
765     print("t_predict_%d_samples: %s\n" % (len(x_test),
766                                         str(pred_finish - pred_start)))
767
768     print(confusion_matrix(y_test, y_pred))
769     print(classification_report(y_test, y_pred))
770
771
772 def list_all_data_files(filter_row=False, rows=[], location=None,
773                        datadir='data'):
774     """
775     Lists all RangeNet log files on disk.
776     Files should be in this format: *-LOCATION-X-Y-{R, LOS}.csv
777     where X and Y can be 0 for a setup that is not in the controlled
778     grid configuration, and * can be anything you want.
779
780     :param filter_row: controlled experimental setup data only - if you would
781                       to only list ranges for particular rows
782     :param rows: controlled experimental setup data only - if the filter_row
783                 flag is enabled, only list files from this array of rows
784     :param location: if you would only like to list files from a particular
785                     location
786     :param datadir: the directory where the RangeNet logs are located.
787     :return: a tuple of: list of LOS files, list of NLOS files
788     """
789     los_files = []
790     reflected_files = []
791
792     for file in os.listdir(datadir):
793         if file.endswith(".csv"):
794             file_components = file.split("-")
795             rtype = file_components[-1].rsplit(".", 1)[0]
796             loc = file_components[-4]
797             row = int(file_components[-3])
798             col = int(file_components[-2])
799
800             if filter_row and not (row in rows):
801                 continue
802
803             if location and loc != location:
804                 continue
805
806             if rtype == "LOS":
807                 los_files.append("".join((datadir, "/", file)))
808             elif rtype == "R":
809                 reflected_files.append("".join((datadir, "/", file)))
810
811     return los_files, reflected_files
812
813
814 def list_all_data_files_real_world():
815     """

```

```

816     Lists all data files for the real world experimental setup.
817     :return: a list of all files in the 'data_real_world' folder
818     """
819     return "".join(("data_real_world/", file))
820         for file in os.listdir("data_real_world")
821         if file.endswith(".csv")]
822
823
824 def parse_log_files(path):
825     """
826     Parses a RangeNet log file into Python FullScanInfo and RangeInfo objects.
827     :param path: the path of the log file to parse
828     :return: a tuple of: list of FullScanInfo of the full scans from the file,
829             list of RangeInfo of the range info entries in the file
830     """
831     full_scans = []
832     range_infos = []
833
834     with open(path, 'r') as logfile:
835         reader = csv.reader(logfile, skipinitialspace=True)
836         for row in reader:
837             type_column = 1
838             if row[0].replace('.', '', 1).isdigit():
839                 if row[type_column] == "RcmFullScanInfo":
840                     scn = FullScanInfo(time=float(row[0]), msg_id=int(row[2]),
841                                       src_id=int(row[3]), emb_time=int(row[4]),
842                                       noise=int(row[5]), v_peak=int(row[6]),
843                                       leading_edge_offset=int(row[7]),
844                                       lockspot_offset=int(row[8]),
845                                       scan_start_ps=int(row[9]),
846                                       scan_stop_ps=int(row[10]),
847                                       scan_step_bins=int(row[11]),
848                                       filtering=int(row[12]), ant_id=int(row[13]),
849                                       num_samples=int(row[15]),
850                                       scan_data=np.asarray(row[16:], np.int32))
851                     full_scans.append(scn)
852                 if row[type_column] == "RcmRangeInfo":
853                     rng = RangeInfo( time=float(row[0]), msg_id=int(row[2]),
854                                     resp_id=int(row[3]), status=int(row[4]),
855                                     req_ant_mode=int(row[5]),
856                                     resp_ant_mode=int(row[6]),
857                                     stopwatch_time=int(row[7]),
858                                     precision_range_mm=int(row[8]),
859                                     course_range_mm=int(row[9]),
860                                     filtered_range_mm=int(row[10]),
861                                     precision_range_err_mm=int(row[11]),
862                                     course_range_err_mm=int(row[12]),
863                                     filtered_range_err_mm=int(row[13]),
864                                     filtered_range_vel_mm_per_sec=int(row[14]),
865                                     filtered_range_vel_mm_per_sec_err=int(row[15]),
866                                     range_type=row[16], req_led_flags=int(row[17]),
867                                     resp_led_flags=int(row[18]), noise=int(row[19]),
868                                     v_peak=int(row[20]),
869                                     course_tof_in_bins=int(row[21]),
870                                     embedded_timestamp=int(row[22]),

```

```

871         reserved=int(row[23]))
872         range_infos.append(rng)
873
874     return full_scans, range_infos
875
876
877 def parse_sort_log_files(filter_row=False, rows=[], location=None,
878                          datadir='data'):
879     """
880     Parses and labels RangeNet log files in a directory.
881     Files should be in this format: *-LOCATION-X-Y-{R, LOS}.csv
882         where X and Y can be 0 for a setup that is not in the controlled
883         grid configuration, and * can be anything you want.
884
885     :param filter_row: controlled experimental setup data only - if you would
886                       to only list ranges for particular rows
887     :param rows: controlled experimental setup data only - if the filter_row
888                 flag is enabled, only list files from this array of rows
889     :param location: if you would only like to list files from a particular
890                     location
891     :param datadir: the directory where the RangeNet logs are located.
892     :return: a tuple of: a list of FullScanInfo objects for LOS ranges,
893                    a list of FullScanInfo objects for the NLOS ranges
894
895     """
896     log.info("Parsing_log_files...")
897     # get all data for training
898     range_files_los, range_files_reflected = list_all_data_files(filter_row,
899                                                                    rows, location,
900                                                                    datadir)
901
902     full_ranges_los = []
903     full_ranges_reflected = []
904
905     # parse all RangeNet log files
906     for file_name in range_files_los:
907         log.info("".join(('Importing_file_', file_name)))
908         full_scans, range_infos = parse_log_files(file_name)
909         full_ranges_los.extend(full_scans)
910
911     for file_name in range_files_reflected:
912         log.info("".join(('Importing_file_', file_name)))
913         full_scans, range_infos = parse_log_files(file_name)
914         full_ranges_reflected.extend(full_scans)
915
916     return full_ranges_los, full_ranges_reflected
917
918 def parse_sort_log_files_real_world():
919     """
920     Parses RangeNet log files for the real world experiment.
921     :return: a tuple of: a list of FullScanInfo objects for LOS ranges,
922                    a list of FullScanInfo objects for the NLOS ranges
923
924     """
925     log.info("Parsing_log_files...")
926     # get all data for training

```

```

926     files = list_all_data_files_real_world()
927
928     all_full_scans = dict()
929     all_ranges = dict()
930
931     full_scans_los = []
932     full_scans_reflected = []
933
934     # Parse all RangeNet log files
935     # WARNING: this assumes all message ids are unique. If they're not,
936     # only one of each msg id will survive.
937     for file_name in files:
938         log.info("".join(('Importing_file_', file_name)))
939         full_scans, range_infos = parse_log_files(file_name)
940
941         full_scans_map = { fs.msg_id: fs for fs in full_scans }
942         ranges_map = { rng.msg_id: rng for rng in range_infos }
943
944         all_full_scans.update(full_scans_map)
945         all_ranges.update(ranges_map)
946
947     # label data
948     # if the range is over this value, it will be considered reflected
949     reflected_range_cutoff_mm = 25000
950     for fs_msg_id, fs in all_full_scans.items():
951         range_info = all_ranges.get(fs_msg_id)
952
953         if range_info.precision_range_mm > reflected_range_cutoff_mm:
954             full_scans_reflected.append(fs)
955         else:
956             full_scans_los.append(fs)
957
958     return full_scans_los, full_scans_reflected
959
960
961 def save_data(inputs, labels, file_name='parsed_data.obj'):
962     """
963     Saves parsed inputs to disk.
964     :param inputs: a list of parsed inputs
965     :param labels: a list of labels for the inputs
966     :param file_name: a name for the saved data
967     :return:
968     """
969     log.info("Saving_parsed_data_to_%s..." % file_name)
970     data_file = open(file_name, 'wb')
971     pickle.dump((inputs, labels), data_file)
972     data_file.close()
973
974
975 def load_data(file_name='parsed_data.obj'):
976     """
977     Loads parsed inputs from disk.
978     :param file_name: the file name of the saved data
979     :return: a tuple of: a list of inputs, a list of labels for the inputs
980     """

```

```

981     log.info("Loading_parsed_data_from_%s..." % file_name)
982     data_file = open(file_name, 'rb')
983     inputs, labels = pickle.load(data_file)
984     data_file.close()
985
986     return inputs, labels
987
988
989 def import_preprocess_data():
990     """
991     Parses and labels all input data and saves it to disk.
992     :return: a tuple of: list of inputs, list of labels for inputs,
993                  list of real world experiment inputs,
994                  list of real world experiment labels
995     """
996     full_ranges_los, full_ranges_reflected = parse_sort_log_files()
997     inputs_raw, labels = ranges_to_features(full_ranges_los,
998                                           full_ranges_reflected)
999
1000     # import testing dataset and scale the same way as training
1001     full_ranges_los_rw, full_ranges_reflected_rw = parse_sort_log_files(datadir="data-real-world")
1002     inputs_rw_raw, labels_rw = ranges_to_features(full_ranges_los_rw,
1003                                                  full_ranges_reflected_rw)
1004
1005     # SCALE INPUTS
1006     # use this for UWB samples, scales all together
1007     inputs, inputs_std, inputs_mean = scale_inputs(inputs_raw)
1008     inputs_rw, _, _ = scale_inputs(inputs_rw_raw, inputs_std, inputs_mean)
1009     # inputs_rw, _, _ = scale_inputs(inputs_rw_raw)
1010
1011     # use this for SVM, scales features independently
1012     # scaler = StandardScaler()
1013     # scaler.fit(inputs_raw)
1014     # inputs = scaler.transform(inputs_raw)
1015     # inputs_rw = scaler.transform(inputs_rw_raw)
1016
1017     # SAVE PROCESSED DATA TO DISK
1018     save_data(inputs, labels)
1019     save_data(inputs_rw, labels_rw, file_name='parsed.data-rw.obj')
1020
1021     return inputs, labels, inputs_rw, labels_rw
1022
1023
1024 def data_func_hyperas():
1025     """
1026     Function that loads input data from disk. Required for hyperas
1027     hyper-parameter optimization.
1028     :return: a tuple of: list of training data, list of training labels,
1029                  list of validation data, list of validation labels,
1030                  if the model should be saved, if hyper-parameter
1031                  optimization is being used
1032     """
1033     import pickle
1034     import logging as log
1035

```

```

1036     seed = 812
1037
1038     log.info("Loading_data_for_hyperas_optimization...")
1039
1040     file_name = 'parsed_data.obj'
1041
1042     data_file = open(file_name, 'rb')
1043     inputs, labels = pickle.load(data_file)
1044     data_file.close()
1045
1046     x_train, x_val, \
1047     y_train, y_val = train_test_split(inputs, labels,
1048                                       test_size=0.4, random_state=seed)
1049
1050     save_model = False
1051     hyper_opt = True
1052
1053     return x_train, y_train, x_val, y_val, save_model, hyper_opt
1054
1055
1056 def main():
1057     """
1058     Main function to initiate training and testing of shallow and deep learning
1059     models.
1060     """
1061     print("Script_started_at: %s\n" % str(datetime.datetime.now()))
1062
1063     # reproducibility
1064     np.random.seed(seed)
1065     set_random_seed(seed)
1066
1067     # IMPORT RANGENET LOG FILES
1068     # inputs, labels, inputs_rw, labels_rw = import_preprocess_data()
1069
1070     # LOAD PROCESSED DATA FROM DISK
1071     inputs, labels = load_data()
1072     inputs_rw, labels_rw = load_data(file_name='parsed_data_rw.obj')
1073
1074     # TRAIN MODEL
1075     # log.info("Training...")
1076     # model = train_nn(inputs, labels, kfold=True, save_model=True,
1077     #                  hyper_opt=False, hyper_opt_max_evals=30)
1078     # train_svm(inputs, labels, save_model=True, kfold=True)
1079
1080     # LOAD SAVED MODEL
1081     model = load_model('models/model_mlp.h5',
1082                       custom_objects={'f1_score': f1_score})
1083     model_name = 'mlp'
1084     model.summary()
1085
1086     get_val_time = False # get the validation time of a saved model
1087     test_rw = True # test real world data on a trained model
1088
1089     if get_val_time:
1090         # get the validation time of a saved model

```

```

1091     x_train, x_val, \
1092     y_train, y_val = train_test_split(inputs, labels,
1093                                       test_size=0.2, random_state=seed)
1094
1095     pred_start = time.time()
1096     if model_name == 'conv' or model_name == 'lstm':
1097         score = model.evaluate(np.reshape(x_val, x_val.shape + (1,)), y_val)
1098     else:
1099         score = model.evaluate(x_val, y_val)
1100     pred_finish = time.time()
1101
1102     print("t_predict_%d_samples: %s\n" % (len(x_val),
1103                                           str(pred_finish - pred_start)))
1104
1105     print("TEST_%s: %.6f%s: %.6f%%\n" % (model.metrics_names[0], score[0],
1106                                           model.metrics_names[1],
1107                                           score[1] * 100))
1108
1109     if test_rw:
1110         # test real world data on a trained model
1111         pred_start = time.time()
1112         if model_name == 'conv' or model_name == 'lstm':
1113             score = model.evaluate(np.reshape(inputs_rw, inputs_rw.shape + (1,)),
1114                                   labels_rw)
1115         else:
1116             score = model.evaluate(inputs_rw, labels_rw)
1117         pred_finish = time.time()
1118
1119         print("t_predict_%d_samples: %s\n" % (len(inputs_rw),
1120                                               str(pred_finish - pred_start)))
1121
1122         print("RW_VAL%s: %.6f%s: %.6f%%\n" % (model.metrics_names[0],
1123                                                score[0],
1124                                                model.metrics_names[1],
1125                                                score[1] * 100,
1126                                                model.metrics_names[2],
1127                                                score[2]))
1128
1129     print("Script_finished_at: %s\n" % str(datetime.datetime.now()))
1130
1131
1132 if __name__ == '__main__':
1133     main()

```

Appendix B

HYPER-PARAMETER SEARCH RESULTS

Rank	Mean CV Score	STD CV Score	C	γ
1	0.9997	9.4245×10^{-5}	1000.0	0.001
2	0.9995	0.0002	100.0	0.01
3	0.9995	0.0003	1.0	0.01
4	0.9993	0.0002	10.0	0.001
5	0.9687	0.0010	10.0	0.1
5	0.9687	0.0010	1000.0	0.1
7	0.9335	0.0047	0.01	0.001
8	0.7764	0.0037	0.01	0.1
9	0.7614	0.0029	1.0	1.0
10	0.5001	1.3898×10^{-5}	0.1	10.0
10	0.5001	1.3898×10^{-5}	0.001	1.0
10	0.5001	1.3898×10^{-5}	100.0	100.0
10	0.5001	1.3898×10^{-5}	10.0	100.0
10	0.5001	1.3898×10^{-5}	10.0	1000.0
10	0.5001	1.3898×10^{-5}	1.0	10.0
10	0.5001	1.3898×10^{-5}	0.01	100.0
10	0.5001	1.3898×10^{-5}	1000.0	10.0
10	0.5001	1.3898×10^{-5}	0.001	1000.0
10	0.5001	1.3898×10^{-5}	1000.0	1000.0
10	0.5001	1.3898×10^{-5}	0.01	1.0

Table B.1: SVM Hyper-Parameter Optimization Results

loss	use_regularization	dropout	optimizer	constr	reg_lambda	learning_rate	neurons_hidden	inner_activation	epochs	batch_size
-1.0000	TRUE	0.3013	adam		1.00E-05	1.00E-05	2229	<i>ReLU</i>	100	32
-1.0000	TRUE	0.3126	adam		1.00E-05	1.00E-05	1510	<i>ReLU</i>	100	128
-1.0000	TRUE	0.3045	adam		1.00E-05	1.00E-05	2718	<i>ReLU</i>	100	32
-1.0000	TRUE	0.4185	adam		1.00E-05	1.00E-05	2660	<i>ReLU</i>	100	32
-1.0000	TRUE	0.4291	adam	<i>maznorm</i> (5)	1.00E-05	1.00E-05	905	<i>ReLU</i>	100	128
-1.0000	TRUE	0.4582	sgd	<i>maznorm</i> (5)	1.00E-05	0.1	2039	<i>ReLU</i>	100	128
-0.9999	TRUE	0.0030	adam	<i>maznorm</i> (4)	0.0001	0.001	2469	<i>ReLU</i>	100	32
-0.9999	FALSE	0.2086	adam		1.00E-05	1.00E-05	957	<i>ReLU</i>	100	128
-0.9999	FALSE	0.0060	rmsprop	<i>maznorm</i> (3)	0.0001	0.0001	2028	<i>ReLU</i>	25	64
-0.9999	TRUE	0.1938	rmsprop	<i>maznorm</i> (4)	1.00E-05	0.0001	2241	<i>LeakyReLU</i>	50	128
-0.9998	TRUE	0.0403	rmsprop	<i>maznorm</i> (4)	0.001	0.001	2790	<i>ReLU</i>	50	128
-0.9998	FALSE	0.0687	rmsprop	<i>maznorm</i> (4)	0.1	0.001	1535	<i>ReLU</i>	50	128
-0.9997	FALSE	0.3595	adam		0.01	0.0001	2670	<i>LeakyReLU</i>	25	128
-0.9997	FALSE	0.2296	rmsprop	<i>maznorm</i> (5)	0.0001	0.001	2279	<i>ReLU</i>	25	64
-0.9996	TRUE	0.4225	adam	<i>maznorm</i> (5)	0.0001	1.00E-05	2843	<i>LeakyReLU</i>	50	64
-0.9996	TRUE	0.4887	adam		0.01	1.00E-05	3148	<i>ReLU</i>	100	32
-0.9994	FALSE	0.1595	adam	<i>maznorm</i> (5)	0.1	0.001	894	<i>LeakyReLU</i>	50	128
-0.9991	FALSE	0.2812	rmsprop	<i>maznorm</i> (4)	0.001	0.001	532	<i>LeakyReLU</i>	100	64
-0.9991	FALSE	0.4793	adam	<i>maznorm</i> (4)	1.00E-05	1.00E-05	3077	<i>LeakyReLU</i>	25	64
-0.9989	FALSE	0.1299	rmsprop	<i>maznorm</i> (3)	0.0001	1.00E-06	2570	<i>LeakyReLU</i>	100	32
-0.9873	TRUE	0.1466	sgd	<i>maznorm</i> (4)	0.01	0.0001	2134	<i>ReLU</i>	50	32
-0.9850	TRUE	0.1230	adam		0.1	1.00E-05	1038	<i>ReLU</i>	100	128
-0.9814	FALSE	0.0716	adam	<i>maznorm</i> (4)	0.0001	1.00E-06	1261	<i>LeakyReLU</i>	25	128
-0.7589	FALSE	0.2477	sgd	<i>maznorm</i> (4)	1.00E-05	1.00E-05	2289	<i>LeakyReLU</i>	100	128
-0.6283	FALSE	0.0851	sgd	<i>maznorm</i> (4)	1.00E-05	1.00E-06	960	<i>ReLU</i>	100	64
-0.5992	TRUE	0.2918	adam		1.00E-05	0.01	655	<i>ReLU</i>	100	32
-0.5856	FALSE	0.2047	sgd		0.0001	1.00E-06	1881	<i>ReLU</i>	50	32
-0.5224	TRUE	0.3368	adam		1.00E-05	0.01	1806	<i>ReLU</i>	100	32
-0.5026	TRUE	0.0056	adam	<i>maznorm</i> (3)	0.0001	0.1	2890	<i>ReLU</i>	100	32
-0.5026	TRUE	0.3935	rmsprop	<i>maznorm</i> (3)	0.001	0.1	891	<i>LeakyReLU</i>	100	64

Table B.2: Multi-Layer Perceptron Hyper-Parameter Optimization Results¹

loss	batch_size	epochs	inner_activation	kernel_size	learning_rate	neurons_first	neurons_second	optimizer	strides_first	strides_pool	strides_second
-1.0000	32	50	LeakyReLU	7	0.0001	64	8	adam	2	7	1
-1.0000	32	50	LeakyReLU	7	0.0001	64	8	adam	2	4	1
-0.9999	32	50	LeakyReLU	7	0.0001	64	8	adam	2	4	1
-0.9999	16	50	LeakyReLU	7	0.0001	64	8	adam	2	7	1
-0.9999	32	50	LeakyReLU	7	0.0001	64	8	adam	2	7	1
-0.9999	32	50	LeakyReLU	7	0.0001	64	8	adam	2	7	1
-0.9995	32	50	LeakyReLU	7	0.0001	64	8	adam	3	4	3
-0.9992	16	50	ReLU	6	0.0001	64	32	adam	2	7	1
-0.9992	32	100	LeakyReLU	8	0.01	64	8	sgd	1	2	2
-0.9992	32	50	LeakyReLU	7	0.0001	64	8	adam	2	4	1
-0.9990	32	100	ReLU	6	1e-05	32	32	rmsprop	4	2	4
-0.9990	32	100	ReLU	5	0.1	64	16	sgd	1	8	2
-0.9989	16	50	LeakyReLU	3	0.001	128	32	adam	1	4	4
-0.9987	64	100	ReLU	8	1e-05	128	16	rmsprop	1	8	1
-0.9984	16	25	ReLU	7	0.001	128	16	sgd	2	6	4
-0.9978	16	25	LeakyReLU	3	0.001	128	32	sgd	1	2	1
-0.9975	64	50	LeakyReLU	6	0.01	64	8	adam	2	7	1
-0.9974	32	100	ReLU	5	1e-05	64	16	rmsprop	1	2	3
-0.9972	64	50	ReLU	8	1e-05	32	32	rmsprop	4	2	4
-0.9971	32	50	LeakyReLU	7	0.01	64	8	adam	2	3	1
-0.9971	16	25	LeakyReLU	4	0.001	32	8	sgd	2	2	2
-0.9969	16	100	LeakyReLU	3	1e-05	128	16	adam	4	1	4
-0.9892	16	100	ReLU	8	1e-06	128	16	adam	3	3	2
-0.9852	16	100	ReLU	4	1e-05	128	16	adam	1	7	2
-0.8283	32	25	ReLU	6	0.0001	64	32	rmsprop	4	5	4
-0.7497	16	50	LeakyReLU	3	1e-05	64	32	rmsprop	4	8	1
-0.7493	32	100	LeakyReLU	3	1e-05	32	32	rmsprop	3	8	3
-0.5229	32	100	LeakyReLU	5	1e-06	128	8	sgd	4	1	2
-0.5026	64	100	LeakyReLU	5	0.1	128	16	rmsprop	1	5	2
-0.4171	32	25	ReLU	4	1e-06	128	32	sgd	4	6	3

Table B.3: Convolutional Neural Network Hyper-Parameter Optimization Results¹

loss	use_regularization	dropout	optimizer	constr	reg_lambda	learning_rate	neurons_hidden	inner_activation	epochs	batch_size	lstm_output
-1	TRUE	0.1604	rmsprop	<i>maxnorm</i> (5)	0.0001	0.001	32	LeakyReLU	15	32	256
-1	TRUE	0.0030	rmsprop	<i>maxnorm</i> (4)	0.0001	0.001	32	ReLU	20	8	128
-1	TRUE	0.1226	rmsprop	<i>maxnorm</i> (4)	0.0001	0.001	32	ReLU	10	32	256
-1.0000	FALSE	0.0726	rmsprop	<i>maxnorm</i> (4)	0.001	0.001	128	LeakyReLU	40	16	64
-1.0000	FALSE	0.1595	rmsprop	<i>maxnorm</i> (5)	0.1	0.001	32	LeakyReLU	15	32	256
-0.9999	FALSE	0.0603	rmsprop	<i>maxnorm</i> (3)	0.0001	0.0001	128	ReLU	40	16	64
-0.9998	TRUE	0.2962	rmsprop	<i>maxnorm</i> (5)	0.0001	0.001	32	ReLU	20	32	64
-0.9996	TRUE	0.2856	rmsprop	<i>maxnorm</i> (5)	0.1	0.001	32	LeakyReLU	20	8	128
-0.9925	TRUE	0.0113	rmsprop	<i>maxnorm</i> (5)	0.0001	0.001	32	ReLU	20	8	128
-0.9786	FALSE	0.4010	rmsprop	<i>maxnorm</i> (5)	1.00E-05	1.00E-05	32	ReLU	40	8	128
-0.9669	TRUE	0.1466	rmsprop	<i>maxnorm</i> (4)	0.01	0.0001	64	ReLU	15	8	256
-0.9647	TRUE	0.3306	rmsprop	<i>maxnorm</i> (5)	0.0001	0.001	32	ReLU	20	8	256
-0.9521	TRUE	0.0403	rmsprop	<i>maxnorm</i> (4)	0.001	0.001	128	ReLU	15	32	256
-0.9512	TRUE	0.2096	rmsprop	<i>maxnorm</i> (5)	0.01	0.001	32	LeakyReLU	15	8	128
-0.9459	TRUE	0.1001	rmsprop	<i>maxnorm</i> (4)	0.0001	0.01	32	LeakyReLU	10	32	256
-0.9290	TRUE	0.1938	rmsprop	<i>maxnorm</i> (4)	1.00E-05	0.0001	128	LeakyReLU	15	32	128
-0.8922	FALSE	0.4085	rmsprop	<i>maxnorm</i> (3)	0.0001	0.01	128	LeakyReLU	15	32	256
-0.8239	FALSE	0.4793	rmsprop	<i>maxnorm</i> (4)	1.00E-05	1.00E-05	32	LeakyReLU	10	16	128
-0.8219	FALSE	0.2477	rmsprop	<i>maxnorm</i> (4)	1.00E-05	1.00E-05	64	LeakyReLU	20	32	128
-0.8208	TRUE	0.4146	rmsprop	<i>maxnorm</i> (5)	0.001	1.00E-05	128	ReLU	15	32	128
-0.8187	FALSE	0.3595	rmsprop		0.01	0.0001	32	LeakyReLU	10	32	64
-0.8129	FALSE	0.0945	rmsprop	<i>maxnorm</i> (3)	0.0001	1.00E-06	128	LeakyReLU	40	16	256
-0.8119	FALSE	0.2404	rmsprop		0.0001	1.00E-06	64	ReLU	40	8	128
-0.7918	FALSE	0.1843	rmsprop		1.00E-05	1.00E-06	64	ReLU	40	16	64
-0.5785	FALSE	0.0716	rmsprop	<i>maxnorm</i> (4)	0.0001	1.00E-06	32	LeakyReLU	10	32	128
-0.5026	TRUE	0.0056	rmsprop	<i>maxnorm</i> (3)	0.0001	0.1	32	ReLU	20	8	256
-0.5026	TRUE	0.1278	rmsprop	<i>maxnorm</i> (5)	0.0001	0.01	32	ReLU	20	8	256
-0.5026	TRUE	0.1227	rmsprop	<i>maxnorm</i> (3)	0.001	0.1	32	ReLU	10	32	256
-0.4974	FALSE	0.4766	rmsprop		0.1	0.1	128	LeakyReLU	20	16	256
0	TRUE	0.2190	rmsprop		0.01	0.1	64	LeakyReLU	10	32	256

Table B.4: Long Short Term Memory Hyper-Parameter Optimization Results¹

¹Loss was defined as the negative of the test set accuracy so that the hyperas library would minimize the negative of accuracy (maximize the loss).

Appendix C

NEURAL NETWORK CROSS VALIDATION DATA

Table C.1: MLP 5-Fold Cross Validation Fold 1

epoch	acc	f1 score	loss	val acc	val f1 score	val loss
0	0.8790300166759311	0.8858237679904868	0.3362071834889765	0.9732508858472869	0.4953073517356697	0.1432733517462389
1	0.9788424124513618	0.9780025784673791	0.1196469812467073	0.992843743486417	0.49760809920705945	0.079139119005439147
2	0.9917488882712618	0.991435812803823	0.07187453311158261	0.9964566108524977	0.49872282369170107	0.05610785621057199
3	0.9960568371317399	0.9959064857240119	0.05422316205011201	0.9980546098798027	0.4992855744953907	0.046832304039719885
4	0.9981065869927738	0.9980364773192626	0.04605809537866766	0.9989578267213228	0.49980314831616757	0.04261619097361727
5	0.9985408560311284	0.9985001183603127	0.04122514285358407	0.9986104356284304	0.4997950210470575	0.039778102563565614
6	0.9989924958310172	0.9989298520120002	0.037908020339752976	0.9993052178142152	0.4998748679642448	0.03649587699643792
7	0.9994615063924402	0.9994250042296702	0.03560876677753927	0.9990967831584798	0.4998303117011956	0.0325544379939465
8	0.9994441356309061	0.9994289577835597	0.03428727527847384	0.9994441742513722	0.5000160305807972	0.03403318075560624
9	0.9996699555308505	0.9996795820620008	0.03231555725416255	0.9993746960327937	0.4999912859359378	0.033586321631689134
10	0.9997568093385214	0.9997339241699486	0.030798517421235835	0.9992357395956367	0.49991956628786055	0.03334460101174296
11	0.9997741801000556	0.9997560765890362	0.029745161420499502	0.9991662613770583	0.4998791679467667	0.03311286632478639
12	0.9997568093385214	0.9997539055089012	0.028626202794621226	0.9992357395956367	0.49990902027858186	0.030495962396316877
13	0.999722067815453	0.9997218657427858	0.027798893137225646	0.9993052178142152	0.49995088759484385	0.029967777443176045
14	0.9997568093385214	0.999752623495491	0.02693107556458511	0.9995136524699507	0.500063005584015	0.028734534372436008
15	0.9998436631461923	0.9998482957424357	0.026074726957664218	0.9995831306885291	0.500098296238153	0.027695028122079916
16	0.999808921623124	0.9998082757460004	0.02523661848507602	0.9997220871256861	0.500098296238153	0.026637886641477846
17	0.9998784046692607	0.9998768839390825	0.024229203878367127	0.9995831306885291	0.500063005584015	0.025788570129487477
18	0.9998957754307949	0.999905650021965	0.023241510425278583	0.9996526089071076	0.500063005584015	0.0253652689684324
19	0.9998784046692607	0.9998843631789445	0.022709095219006267	0.9995831306885291	0.500098296238153	0.024811375884131247
20	0.9998262923846581	0.9998132705423419	0.021889070315020426	0.9995831306885291	0.500063005584015	0.02374339295333896
21	0.9998957754307949	0.9998780255055282	0.021029517509088772	0.9995831306885291	0.5000226072429211	0.02347970447949312
22	0.9998784046692607	0.9998870666256343	0.020379212578803117	0.9997220871256861	0.500098296238153	0.021955060225114963
23	0.9998610339077265	0.9998592770782161	0.019852362827849825	0.9997220871256861	0.500098296238153	0.021371259650864255
24	0.9998784046692607	0.9998700508745331	0.019196135754499785	0.9995831306885291	0.500098296238153	0.021314997480426316
25	0.9999131461923291	0.999910619704176	0.01848078509580433	0.9995831306885291	0.500063005584015	0.020303676418511288
26	0.9998784046692607	0.9998836625675945	0.01814998429850481	0.9995831306885291	0.500098296238153	0.01981942915862814
27	0.9998784046692607	0.9998747665660788	0.01742273317303109	0.9997220871256861	0.500098296238153	0.018901079697051085
28	0.9999131461923291	0.9999085245959424	0.016787470429680162	0.9997220871256861	0.500098296238153	0.018541454338766715
29	0.9999305169538633	0.9999280823078865	0.016154493707925363	0.9996526089071076	0.500098296238153	0.017824739817459256
30	0.9998436631461923	0.9998292252379964	0.015742389601202326	0.9995831306885291	0.500098296238153	0.017779513402625673

epoch	acc	f1'score	loss	val'acc	val'f1'score	val'loss
31	0.9990652584769316	0.9990663112162218	0.014972508213006006	0.9997220871256861	0.500098296238153	0.01682745825978384
32	0.9998957754307949	0.9998915061611411	0.0144495241071090823	0.9996526089071076	0.500063005584015	0.01642311657093352
33	0.9999131461923291	0.9999004191262911	0.014013332992961931	0.9997220871256861	0.500098296238153	0.01594285857924225
34	0.9999131461923291	0.9999070935212221	0.013556451138612162	0.9997220871256861	0.500098296238153	0.01579416469816228
35	0.9999478877153974	0.9999421544045856	0.013073438878951304	0.9997220871256861	0.500098296238153	0.014667203072436505
36	0.9999652584769316	0.9999566641763027	0.012487974663934422	0.9997220871256861	0.500098296238153	0.014336426015266458
37	0.9998784046692607	0.9998516313495605	0.012208039643529754	0.9997220871256861	0.500098296238153	0.013976990401419757
38	0.9999305169538633	0.999930305670495958	0.011644277139760972	0.9997220871256861	0.500098296238153	0.013699713972751935
39	0.9998610339077265	0.9998441666215575	0.011412202268259071	0.9997220871256861	0.500098296238153	0.013314192576149726
40	0.9999131461923291	0.9999205693726807	0.010957618759610608	0.9997220871256861	0.500098296238153	0.012872535144597165
41	0.9998957754307949	0.9998891827727504	0.010894162876489031	0.9997220871256861	0.500098296238153	0.012360129129450349
42	0.9998610339077265	0.999738870745304	0.010551078478521509	0.9997220871256861	0.500098296238153	0.012415218552399679
43	0.9998957754307949	0.9998908949070602	0.010217620880384953	0.9997220871256861	0.500098296238153	0.011986833410505158
44	0.9998262923846581	0.9998209375418048	0.010195107955077644	0.9996526089071076	0.500063005584015	0.011981623270689617
45	0.9999652584769316	0.9999689026101554	0.009706325488907387	0.9996526089071076	0.500063005584015	0.011816442604723148
46	0.9999305169538633	0.9999314705950476	0.009515325221470089	0.9996526089071076	0.500063005584015	0.011752439119019593
47	0.9999305169538633	0.999924146943519	0.009241063555209327	0.9996526089071076	0.500098296238153	0.011246618942044356
48	0.9999131461923291	0.9999196289180715	0.008989490635937422	0.9995831306885291	0.500098296238153	0.012058600223441584
49	0.9999652584769316	0.9999681322226066	0.008571040950427707	0.9995831306885291	0.500098296238153	0.011267918934645066
50	0.9999305169538633	0.9999234805907589	0.008374109828514381	0.9995831306885291	0.500098296238153	0.01136746851243102
51	0.9999305169538633	0.9999145866434331	0.007940934811785543	0.9995831306885291	0.5000578978970592	0.010620647994131689
52	0.9999478877153974	0.9999493425789643	0.007631151681209916	0.9996526089071076	0.500098296238153	0.009927589295992719
53	0.9999131461923291	0.9998920996232852	0.00751295213739265	0.9995831306885291	0.500098296238153	0.010366621379678613
54	0.9999652584769316	0.9999649503484178	0.007174523853245976	0.9997220871256861	0.500098296238153	0.009525500317725021
55	0.9999305169538633	0.9999362644452131	0.006978832901232338	0.9995831306885291	0.500098296238153	0.009423278100711722
56	0.9999478877153974	0.9999482558130092	0.006680213877068767	0.9997220871256861	0.500098296238153	0.008984623051740326
57	0.9999478877153974	0.999939969640894	0.006434811283836867	0.9996526089071076	0.500063005584015	0.00896518592985617
58	0.9999305169538633	0.999919908586113	0.006262519118842514	0.9997220871256861	0.500098296238153	0.008375546793248728
59	0.9999305169538633	0.999926617405294	0.006090711428990392	0.9996526089071076	0.500098296238153	0.008266699508826162
60	0.9999478877153974	0.9999509714523112	0.005896778958174025	0.9997220871256861	0.500098296238153	0.00831292728983854
61	0.9999652584769316	0.9999658088342159	0.005668988131947521	0.9997220871256861	0.500098296238153	0.007736617627149937
62	0.9999478877153974	0.999945723593003	0.005430114078902333	0.9997220871256861	0.500098296238153	0.00764416562829301
63	0.9999131461923291	0.9999178779198105	0.00532797500710543	0.9997220871256861	0.500098296238153	0.007238872543738025

epoch	acc	f1'score	loss	val'acc	val'f1'score	val'loss
64	0.9999478877153974	0.9999493425789643	0.005032168939428041	0.9996526089071076	0.500063005584015	0.007612887563264528
65	0.9999131461923291	0.9999051940911607	0.004939596673961379	0.9996526089071076	0.500063005584015	0.0073308387851743866
66	0.9999478877153974	0.9999439641791891	0.0047965257044461215	0.9996526089071076	0.500063005584015	0.007521632125004966
67	0.9999131461923291	0.9999149592801423	0.00494881195216551	0.9996526089071076	0.500098296238153	0.006639569728187288
68	0.9999478877153974	0.9999478776763717	0.004616906780236012	0.9996526089071076	0.500098296238153	0.006716666190952885
69	0.9999305169538633	0.9999270772788179	0.004573233987309821	0.9997220871256861	0.500098296238153	0.006807151892916313
70	0.9999478877153974	0.9999510607101839	0.0044159431276059175	0.9997220871256861	0.500098296238153	0.006529312112736112
71	0.9999652584769316	0.9999678158442002	0.0042401151204861705	0.9997220871256861	0.500098296238153	0.006352925938327311
72	0.9999478877153974	0.9999476486341764	0.004145724926296085	0.9995831306885291	0.500098296238153	0.007005119184462382
73	0.9999652584769316	0.9999649503484178	0.004013987574902358	0.9996526089071076	0.500098296238153	0.006346101158682327
74	0.9999478877153974	0.9999466861135897	0.003948393602156106	0.9997220871256861	0.500063005584015	0.006796032747705816
75	0.9999652584769316	0.9999639878278311	0.0037973033873340517	0.9997220871256861	0.500098296238153	0.006148355686797695
76	0.9999305169538633	0.9999168883634316	0.003881134542550119	0.9997220871256861	0.500098296238153	0.005756029314517132
77	0.9999652584769316	0.9999639878278311	0.0036096787386899476	0.9997220871256861	0.500098296238153	0.006022687353680872
78	0.9999131461923291	0.9999151514131801	0.0036437724206519993	0.9997220871256861	0.500098296238153	0.005872763072876941
79	0.9999652584769316	0.9999682362573952	0.0035014760436147467	0.9997220871256861	0.500098296238153	0.005916814610063367
80	0.9998784046692607	0.9998824297222182	0.00355937822955894	0.9997220871256861	0.500098296238153	0.005545921457426947
81	0.9999478877153974	0.9999429247921344	0.0034223633897017113	0.9997220871256861	0.500098296238153	0.005869312185147308
82	0.9999131461923291	0.9999095871754657	0.003581591019859375	0.9997220871256861	0.500098296238153	0.005571117478921126
83	0.9999652584769316	0.9999530490999291	0.003296997954006648	0.9997220871256861	0.500098296238153	0.005594630403054508
84	0.9999652584769316	0.9999635305054788	0.003201809043417775	0.9997220871256861	0.500098296238153	0.005373127861160034
85	0.9999652584769316	0.999962901061876	0.003174664557762879	0.9996526089071076	0.500098296238153	0.005138343168478357
86	0.9999305169538633	0.9999312531822188	0.003261254642732346	0.9997220871256861	0.500098296238153	0.005219051597612353
87	0.9999652584769316	0.9999615789586269	0.0030264360878074032	0.9997220871256861	0.500098296238153	0.00529674931548322
88	0.9999652584769316	0.9999731496813034	0.00294730478764127	0.9996526089071076	0.500098296238153	0.005155241943363023
89	0.9999652584769316	0.9999622742688411	0.0028726987996547087	0.9996526089071076	0.500063005584015	0.005684482856866321
90	0.9999131461923291	0.9999032296227905	0.0028360522026791464	0.9995136524699507	0.500063005584015	0.005648214414093677
91	0.9998784046692607	0.9998837209463517	0.002914303246929024	0.9995831306885291	0.500098296238153	0.005415394548801677
92	0.9999652584769316	0.999962901061876	0.0027629928483476054	0.9995136524699507	0.500063005584015	0.0053798107868143945
93	0.9999478877153974	0.9999401094583487	0.002741723357875308	0.9995136524699507	0.500063005584015	0.004895845280610398
94	0.9999652584769316	0.9999576483652816	0.002663780090628423	0.9995831306885291	0.500098296238153	0.005624873974162329
95	0.9999652584769316	0.9999609210214196	0.0026548952748196583	0.9995136524699507	0.500063005584015	0.005570006532285134
96	0.9998610339077265	0.9998466058397638	0.0029151012949912647	0.9997220871256861	0.500098296238153	0.004965056194630373

epoch	acc	f1'score	loss	val'acc	val'f1'score	val'loss
97	0.9998957754307949	0.9998890381180093	0.002767813444787791	0.9997220871256861	0.500098296238153	0.00483888038846593
98	0.9999478877153974	0.9999436478007827	0.0026300760657615076	0.9997220871256861	0.500098296238153	0.004927141666222116
99	0.9999478877153974	0.9999449699040318	0.002607591647195064	0.9997220871256861	0.500098296238153	0.00481011046245249

Table C.2: MLP 5-Fold Cross Validation Fold 2

epoch	acc	f1'score	loss	val'acc	val'f1'score	val'loss
0	0.8853897062655248	0.8918829124558908	0.3286355738437398	0.976236798221234	0.49617827992229874	0.13742434122020367
1	0.9798328961767618	0.9792374551247572	0.11774716405456333	0.9958310172317955	0.49903473687079164	0.0727777143007454
2	0.9922180340113603	0.9919438847384501	0.07185319723798989	0.9974291272929405	0.4991871519443921	0.05044391734309167
3	0.9962479806840487	0.9961211488084946	0.05431758967988754	0.9986798221234019	0.4997362971239583	0.042928370901317715
4	0.9981239903420244	0.9980396628276286	0.045900353181340205	0.9994441356309061	0.5001267981303937	0.03804072000918751
5	0.9988882905730515	0.9988495457389612	0.041007629981384734	0.9994441356309061	0.5000562119179133	0.03558207148600115
6	0.9992356997689729	0.9992419839874431	0.03742160601658687	0.9995831017231795	0.5000562119179133	0.03366988176700471
7	0.999253070228769	0.9992312356938615	0.03569852991889001	0.9998610339077265	0.5002036308036241	0.03243573445033669
8	0.9994094043669336	0.9993214138344894	0.03409292767223205	0.9995831017231795	0.5000573503368095	0.03138255794895497
9	0.9994788862061179	0.999445243283464	0.032541115504609065	0.999722067815453	0.5001683376973838	0.030633913751706075
10	0.9997220726432628	0.9997419142808397	0.03124199893446122	0.9997915508615898	0.5002389239098642	0.029377541553980778
11	0.9997915544824472	0.9997679614557324	0.02977942352999205	0.9995831017231795	0.5000977514849032	0.02878983674975751
12	0.9997915544824472	0.9997677633292265	0.028849966250012262	0.9996525847693163	0.5001683376973838	0.028075937869202237
13	0.9997568135628551	0.9997471948523037	0.027713912130071263	0.999722067815453	0.5002036308036241	0.02718508353103791
14	0.9997915544824472	0.9997645843295938	0.02662444770540376	0.9996525847693163	0.5001683376973838	0.026346882358980285
15	0.9998957772412236	0.9998782396328952	0.025570940757747372	0.9996525847693163	0.5001683376973838	0.02549572536652454
16	0.9997915544824472	0.9997602544371496	0.025118128580427987	0.9995136186770428	0.5002036308036241	0.024884001498398946
17	0.9998784067814275	0.9998562915231278	0.024015317318606445	0.9995831017231795	0.5001330445911435	0.023812842513411293
18	0.9997915544824472	0.9997959124010536	0.02349278471890142	0.9996525847693163	0.5001720528658262	0.023942945993041052
19	0.9998784067814275	0.9998739587388671	0.02269270143878571	0.9996525847693163	0.5001683376973838	0.023220667287318027
20	0.9998610363216315	0.999851733619544	0.022125747971497228	0.9996525847693163	0.5001683376973838	0.022304182813159622
21	0.9998436658618354	0.9998366024168696	0.021433869463069393	0.999722067815453	0.5002036308036241	0.02138322679374601
22	0.9998089249422433	0.9998209215322073	0.02094950962988677	0.999722067815453	0.5002036308036241	0.020977063073448966
23	0.9998784067814275	0.9998443762348145	0.02022182408553109	0.999722067815453	0.5002036308036241	0.020355009393618465
24	0.9998784067814275	0.9998761095710605	0.0194829268241776	0.9996525847693163	0.5001683376973838	0.01975687538760778

epoch	acc	f1'score	loss	val'acc	val'f1'score	val'loss
25	0.9998610363216315	0.9998185308971583	0.018967494916452776	0.9996525847693163	0.5001683376973838	0.019472629219094803
26	0.9998957772412236	0.9998739937288611	0.018238263484499858	0.999722067815453	0.5002036308036241	0.018955076119555506
27	0.9998436658618354	0.9998443666898299	0.017773616982751645	0.9995831017231795	0.5001330445911435	0.018970330344671472
28	0.9999131477010197	0.9999139621370057	0.017180533271798845	0.9996525847693163	0.5001683376973838	0.018666130462128165
29	0.9999131477010197	0.999900736003101	0.016569562998236424	0.9996525847693163	0.5001683376973838	0.01815910770838926
30	0.9998784067814275	0.9998699976334025	0.01606592250190563	0.9996525847693163	0.5001683376973838	0.01787230822663048
31	0.9998957772412236	0.9998779655358077	0.015612133748315049	0.999722067815453	0.5002036308036241	0.016982004868530312
32	0.9998436658618354	0.9998130266910219	0.015154217160630826	0.999722067815453	0.5002036308036241	0.016948199568480964
33	0.9998957772412236	0.9998916146929946	0.014640339579615103	0.999722067815453	0.5002036308036241	0.016390932711972967
34	0.9999305181608157	0.9999244528361041	0.014145509256571069	0.999722067815453	0.5002036308036241	0.015950292631002217
35	0.9998784067814275	0.9998802678090855	0.013818357398633487	0.999722067815453	0.5002036308036241	0.015374181105684346
36	0.9999305181608157	0.9999169498745697	0.01317012512635103	0.9996525847693163	0.5001683376973838	0.015034843964596435
37	0.9998957772412236	0.9998992131095521	0.012651640181516315	0.9996525847693163	0.5001683376973838	0.014314796186449331
38	0.9999305181608157	0.9999134349844534	0.012107646531943113	0.999722067815453	0.5002036308036241	0.013401708668154799
39	0.9998784067814275	0.9998475951610382	0.011692681960029289	0.9996525847693163	0.5001683376973838	0.01307772954003373
40	0.9998784067814275	0.9998585375753176	0.011356120646484846	0.9996525847693163	0.5001683376973838	0.012646048784819227
41	0.9999305181608157	0.9999053661996675	0.010935967818853672	0.9996525847693163	0.5001683376973838	0.012299662566701858
42	0.9999131477010197	0.9999093433407891	0.01068839696411238	0.9996525847693163	0.5001683376973838	0.012276098242918938
43	0.9999478886206118	0.9999349565921285	0.010213664809004762	0.9996525847693163	0.5001683376973838	0.011433277497603337
44	0.9999478886206118	0.9999384099570071	0.009775864743246968	0.999722067815453	0.5002073459720665	0.011066521479623427
45	0.9999131477010197	0.9999010502856197	0.009393924983411244	0.999722067815453	0.5002036308036241	0.010349148189676942
46	0.9999478886206118	0.9999513935019022	0.008904955356818902	0.999722067815453	0.5002036308036241	0.010247965851149472
47	0.9998610363216315	0.9998438400342504	0.008825739654706635	0.9996525847693163	0.5001720528658262	0.0099401626002878
48	0.9999131477010197	0.9998851688589564	0.008373013476879217	0.9997915508615898	0.5002426390783067	0.009303359607811304
49	0.9999478886206118	0.9999402219831429	0.008117835840453337	0.9996525847693163	0.5001683376973838	0.009706443288529695
50	0.9998262954020393	0.999795976944366	0.008097115466636976	0.999722067815453	0.5002036308036241	0.009474911448753348
51	0.9999305181608157	0.999936604948542	0.007958444052884199	0.999722067815453	0.5002036308036241	0.00931442250964077
52	0.9998784067814275	0.9998807790947444	0.007848628100405933	0.999722067815453	0.5002036308036241	0.00925270712294053
53	0.9999131477010197	0.9998957331762983	0.007520239302206566	0.999722067815453	0.5002036308036241	0.008821736461068274
54	0.9999305181608157	0.9999053487693804	0.0072858422706912965	0.999722067815453	0.5002073459720665	0.009142720452165557
55	0.9998957772412236	0.9998730416614235	0.0070800918105418	0.9996525847693163	0.5001683376973838	0.009171664080126474
56	0.9998784067814275	0.9998676939354691	0.007185214318410573	0.9997915508615898	0.5002426390783067	0.0090478417904154
57	0.9999478886206118	0.9999380110834987	0.006714310656822609	0.9996525847693163	0.5001683376973838	0.008394061152892122

epoch	acc	f1'score	loss	val'acc	val'f1'score	val'loss
58	0.9998957772412236	0.9999049563596251	0.006667435209679901	0.999722067815453	0.5002036308306241	0.008320812480974571
59	0.9999478886206118	0.9999398208266655	0.006328574715462024	0.9996525847693163	0.5001683376973838	0.008199639879570878
60	0.9999305181608157	0.9999237583331243	0.006148093474534844	0.9996525847693163	0.5001683376973838	0.00819395913593906
61	0.9999305181608157	0.9999129554056616	0.005917235936098328	0.9996525847693163	0.5001683376973838	0.007763147323323479
62	0.9999305181608157	0.9999293224741853	0.005767705277352697	0.9996525847693163	0.5001683376973838	0.0075984720027839365
63	0.9999131477010197	0.999911889693995	0.005606881864067367	0.9997915508615898	0.5002426390783067	0.007533914184900805
64	0.9999305181608157	0.9999051255654178	0.005426920368540382	0.9996525847693163	0.5001683376973838	0.00734081692950155
65	0.9999305181608157	0.9999227889410064	0.005251359693334212	0.9996525847693163	0.5001683376973838	0.007572298845676928
66	0.9999305181608157	0.9999133044462537	0.005080152364131378	0.9997915508615898	0.5002426390783067	0.007359700890909208
67	0.9999305181608157	0.99993333967900632	0.004914891848118643	0.9996525847693163	0.5001683376973838	0.007099018364123847
68	0.9999131477010197	0.9999147568959732	0.004846539601501328	0.9996525847693163	0.5001683376973838	0.006761803791825642
69	0.9999131477010197	0.9999087922610652	0.004749022362778889	0.999722067815453	0.5002036308306241	0.00646921700558285
70	0.9999305181608157	0.9999269968069464	0.004594404373829845	0.9996525847693163	0.5001683376973838	0.006611327269529223
71	0.9999478886206118	0.9999347328849809	0.004450950764577481	0.9996525847693163	0.5001683376973838	0.006656910566791163
72	0.9998784067814275	0.9998701455656515	0.004451165502877938	0.9996525847693163	0.5001683376973838	0.006534023416317117
73	0.9999478886206118	0.9999316407230693	0.0042630343568715	0.9996525847693163	0.5001683376973838	0.00674994438098652
74	0.9998957772412236	0.999894152449723	0.004331923738234841	0.9996525847693163	0.5001683376973838	0.0064817296941322025
75	0.9999305181608157	0.9999110076034395	0.004237580241554198	0.9996525847693163	0.5001683376973838	0.00609644956764076
76	0.9999305181608157	0.9999266418689432	0.004095526104227063	0.9996525847693163	0.5001683376973838	0.00630064461335722
77	0.9998957772412236	0.9998904712910132	0.004297994982320681	0.9996525847693163	0.5001683376973838	0.006192143917577615
78	0.9998610363216315	0.9998481945434161	0.004473575139687276	0.999722067815453	0.5002073459720665	0.005914500591993796
79	0.9998957772412236	0.9998758612865352	0.004067076539571286	0.999722067815453	0.5002073459720665	0.005951641182471616
80	0.9999131477010197	0.999868685064972	0.003994202808145006	0.9997915508615898	0.5002426390783067	0.005903547460688542
81	0.9999478886206118	0.9999429908507615	0.0038690404476190137	0.999722067815453	0.5002073459720665	0.0060908065483972155
82	0.9999478886206118	0.9999460576039013	0.003775885575216748	0.9995831017231795	0.5001330445911435	0.006339745187076878
83	0.9999478886206118	0.9999358902053079	0.003689267248650762	0.9996525847693163	0.5001683376973838	0.006066376391773409
84	0.9999305181608157	0.9999369116801795	0.003620573523416173	0.999722067815453	0.5002073459720665	0.006151717123298545
85	0.9999305181608157	0.9999424188018899	0.003528494737594335	0.9996525847693163	0.5001683376973838	0.006226234940947626
86	0.9999131477010197	0.9998911315391045	0.0035522312368536854	0.999722067815453	0.5002073459720665	0.005879747318678544
87	0.9999131477010197	0.9999031430082284	0.003572125252164847	0.9996525847693163	0.5001683376973838	0.00595358384093945804
88	0.9999305181608157	0.9999130145785613	0.0034526369178802876	0.999722067815453	0.5002073459720665	0.006021178601617164
89	0.9999305181608157	0.9999138922988617	0.0034666762830080854	0.9996525847693163	0.5001683376973838	0.0059137374536445434
90	0.9999131477010197	0.9999142944130472	0.0034618035999732207	0.9996525847693163	0.5001683376973838	0.005466661450679333

epoch	acc	f1'score	loss	val'acc	val'f1'score	val'loss
91	0.9999478886206118	0.9999523270488185	0.0032944668655659117	0.9997915508615898	0.5002426390783067	0.0055846323255551
92	0.9999478886206118	0.9999513927067456	0.0031858029231377266	0.9996525847693163	0.5001683376973838	0.005598406406148456
93	0.9999478886206118	0.999933793973864	0.003128944756994601	0.9996525847693163	0.5001683376973838	0.004948694387056335
94	0.9998784067814275	0.9998563925742722	0.003333759511961538	0.999722067815453	0.5002036308036241	0.005136249679606149
95	0.9999478886206118	0.9999385660606041	0.0030383113701749767	0.999722067815453	0.5002036308036241	0.005120764271391886
96	0.9998784067814275	0.9998743236825901	0.0031555643970506467	0.999722067815453	0.5002036308036241	0.0052762843035327155
97	0.9999305181608157	0.9999262625792709	0.003010926183411523	0.9996525847693163	0.5001683376973838	0.005188361155578433
98	0.9999131477010197	0.9999151528839331	0.003016336688606801	0.999722067815453	0.5002036308036241	0.005357090573242269
99	0.9999478886206118	0.9999246109097052	0.0029421435328342077	0.9996525847693163	0.5001683376973838	0.005316985250005976

Table C.3: MLP 5-Fold Cross Validation Fold 3

epoch	acc	f1'score	loss	val'acc	val'f1'score	val'loss
0	0.8799701228091508	0.8873830635165507	0.3332860673751697	0.9820733740967204	0.4945420211656283	0.13271253420585655
1	0.9788080390487937	0.9779506157123704	0.11906239623406902	0.9922873818788216	0.49610971304494317	0.06959696404383274
2	0.9922006635515642	0.9918440914323438	0.0718994931759819	0.9977070594774875	0.49926695431385393	0.0507319290574025
3	0.9964564262016016	0.9963490073411145	0.05432819600761207	0.9990272373540856	0.4998308867464601	0.0419513439348567
4	0.9978981743646754	0.9978137642564996	0.04590347981634233	0.9996525847693163	0.5001632296555303	0.037755076829892785
5	0.9987145859750908	0.9986310766136237	0.04125477472942806	0.9997915508615898	0.5002389239098642	0.03558121399209525
6	0.9991141065504003	0.9990925973670463	0.03792324514501499	0.9998610339077265	0.5002389239098642	0.033642803739670726
7	0.9993051816081572	0.9992553134285418	0.03632422872948281	0.9999305169538633	0.5002389239098642	0.032405437402563866
8	0.9996352203442825	0.9996271047777865	0.0337243755055099315	0.9998610339077265	0.5002036308036241	0.031198884363080676
9	0.9996178498844864	0.9996046378946359	0.032507559265640154	0.9998610339077265	0.5002036308036241	0.030230083680396414
10	0.999739443103059	0.9997378883400911	0.03114786345245264	0.9999305169538633	0.5002389239098642	0.028878439441447527
11	0.9996699612638746	0.9996204676060984	0.029964593522412816	0.9999305169538633	0.5002389239098642	0.0278567985226175
12	0.9997568135628551	0.999747951440651	0.028991942201034585	0.9999305169538633	0.5002389239098642	0.026985821320886012
13	0.9998089249422433	0.9997836142811011	0.027835112768280455	0.9998610339077265	0.5002036308036241	0.02611921128794642
14	0.9998089249422433	0.9997889302308192	0.026787543531931298	0.9998610339077265	0.5002389239098642	0.02525547772786762
15	0.9997915544824472	0.9997785174239306	0.025930465084209205	0.9998610339077265	0.5002036308036241	0.024383326050198495
16	0.9997741840226511	0.9997487574312047	0.025275613400352556	0.9999305169538633	0.5002389239098642	0.023653076849815713
17	0.9998610363216315	0.9998199064848511	0.02432881357679656	0.9999305169538633	0.5002389239098642	0.0228797183695636
18	0.9998957772412236	0.9998917773025082	0.02341642915943964	0.9999305169538633	0.5002389239098642	0.021906809420586294

epoch	acc	f1'score	loss	val'acc	val'f1'score	val'loss
19	0.9998436658618354	0.9998351741500556	0.022675441862851673	0.9999305169538633	0.5002389239098642	0.021205484088491638
20	0.9998957772412236	0.9998651671267555	0.021746697788175607	0.9999305169538633	0.5002389239098642	0.02038858371220939
21	0.9998610363216315	0.9998638270529491	0.021155081054940864	0.9999305169538633	0.5002389239098642	0.019805906757737215
22	0.9999305181608157	0.99991956666322613	0.020313281289463473	0.9999305169538633	0.5002389239098642	0.01905987031339539
23	0.9998957772412236	0.9998929186834667	0.019682934864182686	0.99986103339077265	0.5002036308036241	0.018455618346371938
24	0.9999478886206118	0.9999513927067456	0.01888325809733566	0.99986103339077265	0.5002036308036241	0.017837191537942006
25	0.9998610363216315	0.9998556879299404	0.018449504965271344	0.9999305169538633	0.5002389239098642	0.016981361158453074
26	0.9998957772412236	0.9998724044428466	0.017646321807611293	0.9999305169538633	0.5002389239098642	0.016643215072294153
27	0.9999478886206118	0.9999255734135724	0.017037642575222816	0.9999305169538633	0.5002389239098642	0.015966456274752024
28	0.9998957772412236	0.9998758549915459	0.01651890728876403	0.9999305169538633	0.5002389239098642	0.015292873682703755
29	0.9998610363216315	0.99984445115439534	0.016130580371011005	0.9999305169538633	0.5002389239098642	0.01477028581656602
30	0.9999131477010197	0.9999008654448543	0.015482483142507877	0.9999305169538633	0.5002389239098642	0.014338651155351996
31	0.9998610363216315	0.999834803775869	0.015206944367830105	0.9999305169538633	0.5002389239098642	0.013910756115100164
32	0.9999131477010197	0.9998974963365441	0.014751583150868964	0.9999305169538633	0.5002389239098642	0.013502351899298447
33	0.9999305181608157	0.9999185787196223	0.014065927108023863	0.99986103339077265	0.5002036308036241	0.013003513763651973
34	0.9998262954020393	0.9998141997757416	0.013937718145866765	0.99986103339077265	0.5002036308036241	0.01266696620960329
35	0.9998957772412236	0.999871249110411	0.01335714632804533	0.99986103339077265	0.5002036308036241	0.01225769744762484
36	0.9999305181608157	0.9999166175985281	0.012818465411342612	0.99986103339077265	0.5002036308036241	0.011881343832445549
37	0.9999305181608157	0.999906452946745	0.012465394101907532	0.9999305169538633	0.5002389239098642	0.01134509779564651
38	0.9999478886206118	0.9999255734135724	0.011832864411390347	0.99986103339077265	0.5002036308036241	0.010791793498746417
39	0.9999478886206118	0.9999305118989579	0.011208542745655725	0.99986103339077265	0.5002036308036241	0.010191488074032116
40	0.9999131477010197	0.999897782427243	0.010838679577426968	0.99986103339077265	0.5002036308036241	0.009769254016186245
41	0.9999305181608157	0.9999002804777949	0.010516164997934981	0.99986103339077265	0.5002036308036241	0.00972328029196086
42	0.9998436658618354	0.9997960558967847	0.010254964319625318	0.99986103339077265	0.5002036308036241	0.009595739657267423
43	0.9998784067814275	0.9998701049163792	0.01004086583624018	0.9999305169538633	0.5002389239098642	0.00898300239887749
44	0.9999131477010197	0.9999099218140701	0.009686040841980643	0.9999305169538633	0.5002389239098642	0.00879048908221483
45	0.9999305181608157	0.999898982725484	0.009447967056796193	0.9999305169538633	0.5002389239098642	0.008491604568735966
46	0.9998957772412236	0.9998849953491717	0.009313873717091796	0.9999305169538633	0.5002389239098642	0.008255216709776612
47	0.9998957772412236	0.9998720876060947	0.00912426074898487	0.9999305169538633	0.5002389239098642	0.008024821460677758
48	0.9998957772412236	0.9998909734655034	0.00884663217060502	0.9999305169538633	0.5002389239098642	0.007843283011788722
49	0.9999305181608157	0.9999040622091954	0.00860566691356659	0.9999305169538633	0.5002389239098642	0.007653912698115953
50	0.9999478886206118	0.9999492783854902	0.00831728881215422	0.9999305169538633	0.5002389239098642	0.0073683711908770115
51	0.9999131477010197	0.999912716358619	0.008023902256168985	0.9999305169538633	0.5002389239098642	0.007065269820135358

epoch	acc	f1'score	loss	val'acc	val'f1'score	val'loss
52	0.999305181608157	0.9999083474734785	0.007688814111864889	0.9999305169538633	0.5002389239098642	0.0067342630610296534
53	0.9998784067814275	0.9998438855900942	0.007526009599961729	0.9999305169538633	0.5002389239098642	0.006466604256717744
54	0.999305181608157	0.9999291585062793	0.007148622430710998	0.9999305169538633	0.5002389239098642	0.006237148062914626
55	0.9999478886206118	0.9999490949693803	0.006839654943352905	0.9999305169538633	0.5002389239098642	0.005897782168860002
56	0.9999131477010197	0.998903574210756	0.006526023996228866	0.9999305169538633	0.5002389239098642	0.00555247212426241
57	0.9999131477010197	0.99990795360283	0.006302860754849312	0.9999305169538633	0.5002389239098642	0.0053582885859118955
58	0.9998784067814275	0.9998706244155496	0.006267390755367735	0.9999305169538633	0.5002389239098642	0.005182701576886292
59	0.9999478886206118	0.9999460576039013	0.005913460224944649	0.9999305169538633	0.5002389239098642	0.005048897498654302
60	0.9999131477010197	0.999903169543472	0.005936623324417929	0.9999305169538633	0.5002389239098642	0.004923412983221369
61	0.9999478886206118	0.9999349277014407	0.005623965878602252	0.9999305169538633	0.5002389239098642	0.004785330728754575
62	0.9998610363216315	0.9998757685814285	0.005682478060287412	0.9999305169538633	0.5002389239098642	0.004674399881186238
63	0.999305181608157	0.9999283780601296	0.005464220400113099	0.9999305169538633	0.5002389239098642	0.004637031885516433
64	0.9998957772412236	0.9998828802327598	0.005418434724480304	0.9999305169538633	0.5002389239098642	0.004480078187421702
65	0.999478886206118	0.999936509565994	0.005213558803497049	0.9999305169538633	0.5002389239098642	0.0043842592954805425
66	0.999478886206118	0.9999355120752388	0.0051076236416279066	0.9999305169538633	0.5002389239098642	0.004243606804513381
67	0.999305181608157	0.9999169609705237	0.004993892624061128	0.9999305169538633	0.5002389239098642	0.004101941506458594
68	0.999478886206118	0.9999362824494057	0.004765827261132061	0.9999305169538633	0.5002389239098642	0.003944218307816366
69	0.999305181608157	0.9999335861035842	0.004701783060816734	0.9998610339077265	0.5002036308036241	0.0039046630278859224
70	0.9999131477010197	0.9998981605573121	0.004575960581255449	0.9999305169538633	0.5002389239098642	0.003668417812833949
71	0.9999305181608157	0.9999178762951025	0.0044498228319839244	0.9999305169538633	0.5002389239098642	0.003591354715825784
72	0.9999478886206118	0.9999350125812959	0.004333062388366413	0.9999305169538633	0.5002389239098642	0.0034913797751524637
73	0.9999478886206118	0.9999180444406865	0.00423187909498957	0.9999305169538633	0.5002389239098642	0.0033987943792799535
74	0.9998262954020393	0.9997775098642986	0.004598378094166246	0.9998610339077265	0.5002036308036241	0.0034515970581215807
75	0.9999131477010197	0.9999077907945273	0.0043235947779626666	0.9999305169538633	0.5002389239098642	0.003352006058415202
76	0.9999478886206118	0.999954895070505	0.004054633476891047	0.9999305169538633	0.5002389239098642	0.003232741578942317
77	0.999305181608157	0.9999063404983568	0.004011548708969314	0.9999305169538633	0.5002389239098642	0.00318245462148661
78	0.9999131477010197	0.9999166177610797	0.004113971728919664	0.9999305169538633	0.5002389239098642	0.0030878565918442988
79	0.9999131477010197	0.9999132668388694	0.003938668172359808	0.9999305169538633	0.5002389239098642	0.0030265817696434666
80	0.999305181608157	0.9999059725727967	0.003948243386786837	0.9999305169538633	0.5002389239098642	0.0030006950318900295
81	0.9999478886206118	0.9999398208266655	0.003732971677007537	0.9999305169538633	0.5002389239098642	0.002889901874592117
82	0.999305181608157	0.9999086520515671	0.003700622064906495	1.0	0.500277932184547	0.002800912291625346
83	0.9999131477010197	0.9998904409094068	0.0036527455893651368	0.9999305169538633	0.5002389239098642	0.002748421246570024
84	0.9999131477010197	0.9999188779604297	0.0036370903748910387	0.9999305169538633	0.5002389239098642	0.0027100887551299063

epoch	acc	f1'score	loss	val'acc	val'f1'score	val'loss
85	0.999305181608157	0.9999274999753823	0.0034884366475096285	0.9999305169538633	0.5002389239098642	0.0026462630885422827
86	0.999305181608157	0.9999171789459179	0.0035306080677593716	0.9999305169538633	0.5002389239098642	0.002590615755431623
87	0.999305181608157	0.9999212330184354	0.003363858511774863	0.9999305169538633	0.5002389239098642	0.002574234262829426
88	0.999478886206118	0.9999387557476039	0.0032883359024940932	0.9999305169538633	0.5002389239098642	0.0024566638193132771
89	0.999305181608157	0.9999080950444067	0.003275114924816988	0.9999305169538633	0.5002389239098642	0.0024175714934891865
90	0.999478886206118	0.9999493664822092	0.003143327767358065	0.9998610339077265	0.5002036308036241	0.0024947698541004223
91	0.9999131477010197	0.9998935140600365	0.003146923439505782	0.9998610339077265	0.5002036308036241	0.0025112998626273066
92	0.9999131477010197	0.9998952669495103	0.003222328548798864	0.9999305169538633	0.5002389239098642	0.0022377295509133986
93	0.9999478886206118	0.9999243837931168	0.003001136940641773	0.9999305169538633	0.5002389239098642	0.0022303570414468304
94	0.9999478886206118	0.9999353838893773	0.0029352387773792996	0.9999305169538633	0.5002389239098642	0.0021327360338526777
95	0.9998957772412236	0.9998692422046584	0.0029988784502465856	0.9999305169538633	0.5002389239098642	0.002120722213883228
96	0.9999478886206118	0.9999475455074561	0.002896465126582969	0.9999305169538633	0.5002389239098642	0.0020873401736713256
97	0.9998957772412236	0.9998968897615197	0.003016869625849354	0.9999305169538633	0.5002389239098642	0.00207690664490953
98	0.9999478886206118	0.9999293165798804	0.002808088348814213	0.9999305169538633	0.5002389239098642	0.002009424772978102
99	0.9999131477010197	0.9999186821199987	0.0028689318784214284	0.9998610339077265	0.5002036308036241	0.0021602530740336074

Table C.4: MLP 5-Fold Cross Validation Fold 4

epoch	acc	f1'score	loss	val'acc	val'f1'score	val'loss
0	0.8811165731556915	0.8885777048753283	0.33248397402520863	0.9779738743746526	0.4932593865948561	0.13712741923272576
1	0.9809967169830985	0.9803777805792144	0.1171075084936617	0.9955530850472485	0.4984013547892038	0.07283812930280464
2	0.9927591478052424	0.992361266418276	0.07127376489619383	0.9970817120622568	0.4988799696832713	0.05230292788457711
3	0.9965259080407859	0.9964102404250814	0.0541336128973864	0.9981934408004447	0.4993847968380871	0.04367617364473776
4	0.9982455835605969	0.9982224359240485	0.04534642584670439	0.9990967204002223	0.4999115683092814	0.03847641219954944
5	0.9989056610328475	0.998887864631089	0.04073129877678191	0.9990967204002223	0.49983326655881416	0.036371281738079804
6	0.9989230314926436	0.9988336658363479	0.0382363121528789	0.9992358684924958	0.4998762752030411	0.034372298238491465
7	0.9992878111483611	0.9992490804976687	0.03591609812729907	0.9993746525847693	0.49994799983441796	0.03287567446870795
8	0.9995657385050982	0.9995264940838826	0.034041269687896655	0.9993746525847693	0.49998329294065824	0.03168557378659651
9	0.99951362712571	0.999506432976887	0.03261100972014442	0.9994441356309061	0.49998329294065824	0.0303977773622664
10	0.9996525908040785	0.9996325853942787	0.031470439195146055	0.9996525847693163	0.5000926434430498	0.029372552880865127
11	0.9997047021834667	0.9996885545056745	0.030102187666741956	0.9997915508615898	0.5001709144469298	0.028394608797216628
12	0.9998262954020393	0.9998349524670369	0.028932531422873192	0.9995136186770428	0.5000562119179133	0.027409323758801334

epoch	acc	f1'score	loss	val'acc	val'f1'score	val'loss
13	0.9997915544824472	0.9998064761882912	0.028096985108685146	0.999722067815453	0.5001279365492901	0.02644096478496809
14	0.9997915544824472	0.9997624739509896	0.02695588093619125	0.9995831017231795	0.5000926434430498	0.025834821215341527
15	0.9997915544824472	0.9997433742907585	0.026134334814565265	0.9996525847693163	0.5000926434430498	0.024806749187472
16	0.9998784067814275	0.9998616085361607	0.02514117242131887	0.9997915508615898	0.5001632296555303	0.023801041461564622
17	0.9998957772412236	0.9998752910267656	0.02410121095432311	0.9996525847693163	0.5001279365492901	0.022958382057143026
18	0.9998610363216315	0.9998414098370572	0.02315586318501138	0.999722067815453	0.5001632296555303	0.021887345850393133
19	0.9997568135628551	0.99975515085428	0.022527395725163792	0.9999305169538633	0.500277932184547	0.021065591280040576
20	0.9999131477010197	0.9998884555060126	0.021591266164661848	0.999722067815453	0.5001632296555303	0.020556497299063198
21	0.9998436658618354	0.9998236925564811	0.02107747028198537	0.9997915508615898	0.5001632296555303	0.019840210806650875
22	0.9998262954020393	0.9997980173160627	0.020450597417284645	0.9998610339077265	0.5002036308036241	0.01911534407507916
23	0.9998262954020393	0.9998072410325998	0.01982953632861129	0.9998610339077265	0.5001985227617706	0.01866401572105552
24	0.9998436658618354	0.9998173252741772	0.019370727018663342	1.0	0.500277932184547	0.018106674206023285
25	0.9998957772412236	0.9998828116805332	0.01864926113481377	1.0	0.500277932184547	0.017357432254988196
26	0.9998957772412236	0.9998936316407047	0.017991803284449052	1.0	0.500277932184547	0.01676874915412262
27	0.9998784067814275	0.999860987813976	0.01732637381463936	0.9997915508615898	0.5001985227617706	0.016203064854573117
28	0.9998957772412236	0.999901208124194	0.016728875661864907	1.0	0.500277932184547	0.015452201314593534
29	0.9998957772412236	0.9998815685557745	0.01613947116099309	0.9997915508615898	0.5001632296555303	0.015037033719327995
30	0.9998957772412236	0.9998923351048251	0.015548600847251844	0.9999305169538633	0.5002389239098642	0.01433115779293319
31	0.9998957772412236	0.9998725754346356	0.014995960153138591	0.9999305169538633	0.5002426390783067	0.013880134740841262
32	0.9999305181608157	0.9999296990802043	0.014421538438095167	0.9999305169538633	0.5002426390783067	0.013470940048567715
33	0.999947886206118	0.9999423144375933	0.01379838365176281	0.9998610339077265	0.5002036308036241	0.012842867043016353
34	0.9998610363216315	0.9998734920813692	0.013512455266468664	0.9999305169538633	0.5002389239098642	0.012343805141978923
35	0.9998957772412236	0.9998801498639706	0.013043381857220432	1.0	0.500277932184547	0.011984927173892348
36	0.9999131477010197	0.999898952876054	0.012681231509894328	0.9998610339077265	0.5001985227617706	0.011625651021660402
37	0.9999305181608157	0.9999279677262204	0.012186424845572415	1.0	0.500277932184547	0.011072921270547044
38	0.9998262954020393	0.9997958374937868	0.012057867415587651	1.0	0.500277932184547	0.010751135273007733
39	0.9998957772412236	0.9998734519953327	0.011646268913185967	0.9999305169538633	0.5002389239098642	0.010494066095844185
40	0.9998436658618354	0.9998331352723836	0.011447084925203093	0.9998610339077265	0.5001985227617706	0.010294131494607112
41	0.9998957772412236	0.9998762326577736	0.011168092062965393	1.0	0.500277932184547	0.009987186051370901
42	0.9998957772412236	0.9999155711031447	0.0108406997180581	1.0	0.500277932184547	0.009758845992021889
43	0.9998610363216315	0.9998490874679795	0.010582710801744885	1.0	0.500277932184547	0.009495898778382078
44	0.9999131477010197	0.9998901932212488	0.010317584284782477	1.0	0.500277932184547	0.009247024997596677
45	0.9999131477010197	0.9998884570300626	0.00993091994495491	1.0	0.500277932184547	0.0089688686056332007

epoch	acc	f1'score	loss	val'acc	val'f1'score	val'loss
46	0.9998784067814275	0.9998611580768715	0.009683476652157415	0.9998610339077265	0.5001985227617706	0.008699738416922231
47	0.9998784067814275	0.9998721611154969	0.009449844348812474	0.9998610339077265	0.5001985227617706	0.008501043046238093
48	0.9998784067814275	0.9998532489559997	0.009451270497198795	1.0	0.500277932184547	0.008294343564238993
49	0.9999131477010197	0.9998855466577102	0.009155150633084327	0.9999305169538633	0.5002389239098642	0.00810639143068801
50	0.9999305181608157	0.999922852484161	0.008883004959067457	0.9999305169538633	0.5002389239098642	0.007901409868173727
51	0.9999478886206118	0.9999346536043531	0.008563728613905813	0.9997915508615898	0.5001632296555303	0.007865284413737081
52	0.9998784067814275	0.9998843980178421	0.008604255187665376	0.9999305169538633	0.5002389239098642	0.0074438269720156535
53	0.9999305181608157	0.9999283780601296	0.0081512120537042555	0.9998610339077265	0.5001985227617706	0.0072706027555405235
54	0.9999131477010197	0.9998956873554024	0.007905655973735133	1.0	0.500277932184547	0.006871474428198664
55	0.9998957772412236	0.999871402744388	0.0076800380029117795	1.0	0.500277932184547	0.006737018522266516
56	0.9999131477010197	0.9998969239563572	0.0074664689384659816	1.0	0.500277932184547	0.0064663305063796675
57	0.9999131477010197	0.9998956681059876	0.007241067963836946	0.9997915508615898	0.5001632296555303	0.0065370104032329976
58	0.9999305181608157	0.9999117888447459	0.007109109571499929	0.9999305169538633	0.5002426390783067	0.0062761328707533525
59	0.9999131477010197	0.9999240180179996	0.006987397899758117	0.9999305169538633	0.5002389239098642	0.005906489573154037
60	0.9998957772412236	0.9998837945934185	0.00822595811457096	0.9999305169538633	0.5002389239098642	0.005794955894046604
61	0.9999305181608157	0.9999125314546966	0.006515224459950652	1.0	0.500277932184547	0.005617519317473336
62	0.9999305181608157	0.9999298429372762	0.0063933348135290756	1.0	0.500277932184547	0.005403714695828384
63	0.9999305181608157	0.999934046234172	0.006164307332932046	1.0	0.500277932184547	0.00519954271432325
64	0.9999131477010197	0.9999063074631226	0.006013060593308631	1.0	0.500277932184547	0.005009884672449605
65	0.9999305181608157	0.9999305590751403	0.005757359566173824	1.0	0.500277932184547	0.004827689774968082
66	0.9998957772412236	0.9998953460644805	0.00582188761316051	1.0	0.500277932184547	0.004667819234303894
67	0.9999131477010197	0.9999163902800444	0.005502319107291167	0.9999305169538633	0.5002389239098642	0.004579292276119394
68	0.9999478886206118	0.9999402219831429	0.005309153490027539	1.0	0.500277932184547	0.004426316633041929
69	0.9999131477010197	0.999908959310203	0.005390141338963365	1.0	0.500277932184547	0.004283356259764897
70	0.9999305181608157	0.9999073209263778	0.005088482519019005	1.0	0.500277932184547	0.004237747960581572
71	0.9999305181608157	0.9999289394075194	0.004987498583061889	1.0	0.500277932184547	0.004085820631847291
72	0.9999305181608157	0.9999074154506122	0.0048256773252153265	1.0	0.500277932184547	0.003913238811295784
73	0.9998957772412236	0.999906478620569	0.004873048506402975	1.0	0.500277932184547	0.003816544539680815
74	0.9998610363216315	0.9998525795967397	0.0048473371561029025	0.9998610339077265	0.5001985227617706	0.003957959713042611
75	0.9999305181608157	0.9999261467514674	0.004605137496130529	0.9999305169538633	0.5002389239098642	0.0037109876678466567
76	0.9999478886206118	0.9999332103952223	0.004444444216591096	1.0	0.500277932184547	0.003573108417887433
77	0.9999305181608157	0.9999272250831381	0.00436956406513363	1.0	0.500277932184547	0.0034976964037329076
78	0.9999478886206118	0.9999478929246036	0.004312096521104733	0.9999305169538633	0.5002426390783067	0.0034884602991577286

epoch	acc	f1'score	loss	val'acc	val'f1'score	val'loss
79	0.9999131477010197	0.9998984877310992	0.004340916438742517	1.0	0.500277932184547	0.0033628953173766955
80	0.9999478886206118	0.9999342320719892	0.004115946941523497	1.0	0.500277932184547	0.003275514418272723
81	0.9999305181608157	0.9999244536312606	0.0040756699629014735	1.0	0.500277932184547	0.0031488762954648597
82	0.9998784067814275	0.9998631714826146	0.004164681228555481	1.0	0.500277932184547	0.003122037111482831
83	0.9999478886206118	0.9999286871472113	0.0039124807955318594	1.0	0.500277932184547	0.0030685702005940688
84	0.9999305181608157	0.9999283780601296	0.003917607824025314	1.0	0.500277932184547	0.0029881389629408477
85	0.9999131477010197	0.9999247962774688	0.0037923844619151726	1.0	0.500277932184547	0.0029225383085116387
86	0.9999131477010197	0.9998962320707678	0.003848254558841226	1.0	0.500277932184547	0.0028609606512967683
87	0.9998957772412236	0.9998926795732671	0.003822660368909717	1.0	0.500277932184547	0.002834220867233352
88	0.9999478886206118	0.9999299937882052	0.0036317996818682097	1.0	0.500277932184547	0.0028108830808073267
89	0.9999305181608157	0.9999093739211846	0.003579397575038251	1.0	0.500277932184547	0.0027172964018639624
90	0.9999305181608157	0.9999333578273925	0.003593990045042114	1.0	0.500277932184547	0.0026786871693962114
91	0.9998957772412236	0.999908135627416	0.0037493785555878795	1.0	0.5001985227617706	0.002741882165458018
92	0.999478886206118	0.9999349847539228	0.0034250565435208755	1.0	0.500277932184547	0.002580577211496961
93	0.999305181608157	0.999904356715301	0.003375783895338399	1.0	0.500277932184547	0.002492852044525272
94	0.999131477010197	0.999894906885557	0.003374103456747744	1.0	0.5001632296555303	0.0027179767978452703
95	0.9998957772412236	0.9998710472762805	0.0034591211947980357	1.0	0.500277932184547	0.002436899231268448
96	0.999305181608157	0.9999181567865741	0.0032629621514513145	1.0	0.500277932184547	0.002417073388371188
97	0.9999478886206118	0.9999292715210094	0.0031879497586956517	1.0	0.500277932184547	0.002348292397160193
98	0.9999131477010197	0.9999141253097552	0.0031643807285378638	1.0	0.500277932184547	0.002265841001256464
99	0.9999305181608157	0.9999379984272571	0.0032789211765533286	1.0	0.500277932184547	0.002227413365689871

Table C.5: MLP 5-Fold Cross Validation Fold 5

epoch	acc	f1'score	loss	val'acc	val'f1'score	val'loss
0	0.8826625440775417	0.8887073765996825	0.3308260289684551	0.978390772651473	0.495530999455603	0.1395842313948706
1	0.9787211867498132	0.9779535962241565	0.11979754297459588	0.9953446359088383	0.4982392647453253	0.07472140147967296
2	0.9927044068856503	0.9924490198315181	0.071686094997605	0.9964563646470261	0.4985549101370981	0.053833789551933985
3	0.9965780194201741	0.9963873572774792	0.05366179776842125	0.9985408560311284	0.4995580151586018	0.04467829057181657
4	0.9982108426410047	0.9981751299376129	0.04506514595194816	0.9994441356309061	0.5000415069650053	0.040340437155720124
5	0.9988361791936633	0.9987837269209314	0.04007082054084357	0.9995136186770428	0.5000849279050631	0.037728005810006585
6	0.9992183293091769	0.9991789076312084	0.03698115493473841	0.999722067815453	0.5001267981303937	0.035892111106903574

epoch	acc	f1'score	loss	val'acc	val'f1'score	val'loss
7	0.9994615157463218	0.9994048806219498	0.03486272151788837	0.999722067815453	0.5001267981303937	0.03433838859407553
8	0.99951362712571	0.9994695904616503	0.03332127273051931	0.999722067815453	0.5001267981303937	0.033096817403352546
9	0.9996352203442825	0.9996442969206211	0.03183260803225612	0.999722067815453	0.5001267981303937	0.032104712648481845
10	0.9997568135628551	0.9997585734068425	0.030598662791921506	0.9998610339077265	0.5002062075531701	0.031028856246371347
11	0.9998262954020393	0.9997996689556029	0.02925690394651047	0.9998610339077265	0.5002062075531701	0.030021621580088118
12	0.9997568135628551	0.9997546118406426	0.028399212208644883	0.9997915508615898	0.5001671992784874	0.029200280448623734
13	0.999739443103059	0.9997404346632716	0.02748951882445038	0.9997915508615898	0.5001671992784874	0.028344296809366638
14	0.9999305181608157	0.9998971082338868	0.026281202062593004	0.9997915508615898	0.5001671992784874	0.02735537904552316
15	0.9998610363216315	0.9998555721021369	0.025229286902801674	0.9997915508615898	0.5001671992784874	0.026364760070608482
16	0.9999305181608157	0.9999271092553346	0.024398199406457703	0.9998610339077265	0.5002062075531701	0.025495811173482163
17	0.9998610363216315	0.9998537104087424	0.023545519502186352	0.9998610339077265	0.5002062075531701	0.024679004381917596
18	0.9999305181608157	0.9999182016166305	0.0225542270495599	0.9997915508615898	0.5001671992784874	0.023919567181053196
19	0.9997741840226511	0.9997690961441233	0.02237924772386507	0.9998610339077265	0.5002062075531701	0.023384387968827646
20	0.9998784067814275	0.9998789642161916	0.021506684667118803	0.9998610339077265	0.5002062075531701	0.022893753190581304
21	0.9998957772412236	0.9998674092362941	0.020988137854859847	0.9998610339077265	0.5002062075531701	0.022331169810044627
22	0.9999131477010197	0.9999060546033409	0.020421445060115437	0.9998610339077265	0.5002062075531701	0.021782518263539715
23	0.9999478886206118	0.9999328314699967	0.019536476445912405	0.9998610339077265	0.5002062075531701	0.021024955665405888
24	0.999826295402039	0.9999634620928773	0.018619348347820166	0.9998610339077265	0.5002062075531701	0.020033465332872938
25	0.9998957772412236	0.9998837741844004	0.017871145914592497	0.9998610339077265	0.5002062075531701	0.01918931845254604
26	0.999478886206118	0.9999286220769006	0.017140141362973885	0.9998610339077265	0.5002062075531701	0.018754418742630202
27	0.999652590804079	0.9999598349500558	0.01659996712587409	0.9997915508615898	0.5001671992784874	0.018007604734534156
28	0.999478886206118	0.9999347328849809	0.015720505434770275	0.9998610339077265	0.5002062075531701	0.017235803894653064
29	0.999652590804079	0.9999517718007349	0.014930112176440672	0.9998610339077265	0.5002062075531701	0.016444826604236226
30	0.999826295402039	0.9999831558975997	0.014153892131072142	0.9998610339077265	0.5002062075531701	0.015729902800340298
31	0.9998610363216315	0.9998312965060029	0.01361747037756667	0.9997915508615898	0.5001671992784874	0.01547211858590422
32	0.9997915544824472	0.9997737746495561	0.01369297806478645	0.9998610339077265	0.5002062075531701	0.014991709087592751
33	0.9999305181608157	0.9998964946712171	0.013066921369068046	0.9997915508615898	0.5001709144469298	0.014816081939806635
34	0.9999305181608157	0.999936177452504	0.012802017586738226	0.9998610339077265	0.5002062075531701	0.01443440543840732
35	0.999652590804079	0.9999488170984051	0.012327246156863507	0.9997915508615898	0.5001709144469298	0.014162800891356974
36	0.9998957772412236	0.99988683580309	0.011873110900089554	0.9998610339077265	0.5002062075531701	0.013431461004559997
37	0.999478886206118	0.9999448210029465	0.01133410342754794	0.9998610339077265	0.5002062075531701	0.01303409796421431
38	0.999478886206118	0.9999543608273421	0.010860759882698131	0.9998610339077265	0.5002062075531701	0.012485563893776062
39	0.999652590804079	0.9999468378874995	0.010436088701688395	0.9998610339077265	0.5002062075531701	0.012096394269130275

epoch	acc	f1'score	loss	val'acc	val'f1'score	val'loss
40	0.9999305181608157	0.9999360532092937	0.01000209632015233	0.9998610339077265	0.5002062075531701	0.011714257285404761
41	0.9999131477010197	0.9999165137280982	0.00969808899763316	0.9998610339077265	0.5002062075531701	0.01138661331132162
42	0.9999826295402039	0.9999584619165964	0.009310855797017919	0.9998610339077265	0.5002062075531701	0.011099347998991219
43	0.9999652590804079	0.9999442946455507	0.008970395554358655	0.9998610339077265	0.5002062075531701	0.0106767154111837
44	0.9999652590804079	0.9999489413416154	0.008653645072837106	0.9998610339077265	0.5002062075531701	0.010370635523307052
45	0.9999826295402039	0.9999676064156627	0.008295116841044494	0.9998610339077265	0.5002062075531701	0.010094325629690908
46	0.9999305181608157	0.9999364165627036	0.008152617120638382	0.9998610339077265	0.5002062075531701	0.009824062312946445
47	0.9999305181608157	0.9999128055517843	0.007874976266252836	0.9998610339077265	0.5002062075531701	0.0096224998421615
48	0.9999305181608157	0.9999151296949732	0.007824942273201997	0.9998610339077265	0.5002062075531701	0.009430314922573799
49	0.9998784067814275	0.9998666559880185	0.0076485180468731	0.9998610339077265	0.5002062075531701	0.009258441861021049
50	0.9998957772412236	0.9998935308214809	0.0074757226443124155	0.9998610339077265	0.5002062075531701	0.009161872377183253
51	0.9999478886206118	0.9999237073799484	0.007255939062339611	0.9998610339077265	0.5002062075531701	0.00905379373269604
52	0.999826295402039	0.9999735306932699	0.006955381952618046	0.9998610339077265	0.5002062075531701	0.00880404634441582
53	0.9999305181608157	0.9999328916006044	0.006778346976514866	0.9998610339077265	0.5002062075531701	0.008501608693468
54	0.999826295402039	0.9999634620928773	0.006467303222973859	0.9998610339077265	0.5002062075531701	0.008319309261886858
55	0.9999478886206118	0.9999319079288003	0.006234697087973848	0.9998610339077265	0.5002062075531701	0.008025188512816453
56	0.9999478886206118	0.999924536469941	0.006019803469575124	0.9998610339077265	0.5002062075531701	0.007802676382821971
57	0.999826295402039	0.9999831558975997	0.005737125934465878	0.9998610339077265	0.5002062075531701	0.007571281937197919
58	0.9999305181608157	0.9999299477322826	0.005623536793246119	0.9998610339077265	0.5002062075531701	0.007438244388696053
59	0.999478886206118	0.9999481076500025	0.005532182480737855	0.9998610339077265	0.5002062075531701	0.007284657002321593
60	0.999826295402039	0.9999841184014668	0.005263809337919848	0.9998610339077265	0.5002062075531701	0.007126247620840596
61	0.999826295402039	0.9999676064156627	0.005072122543876684	0.9998610339077265	0.5002062075531701	0.006907881352493437
62	0.999131477010197	0.9998923681400593	0.005087637468729473	0.9998610339077265	0.5002062075531701	0.006781443863953896
63	0.9999478886206118	0.9999553099129426	0.00489211769180486	0.9998610339077265	0.5002062075531701	0.00670184122875805
64	0.999826295402039	0.9999808325495673	0.004701240989409109	0.9998610339077265	0.5002062075531701	0.006608625020512927
65	0.999826295402039	0.9999841184014668	0.004587679469242611	0.9997915508615898	0.5001709144469298	0.006878317114775415
66	0.999826295402039	0.9999820691505221	0.00440656877423543	0.9998610339077265	0.5002062075531701	0.006311007613070793
67	0.9999652590804079	0.9999658094250261	0.004265367973156283	0.9997915508615898	0.5001709144469298	0.006319045061747719
68	0.9999652590804079	0.9999466179935831	0.004061931078909099	0.9998610339077265	0.5002062075531701	0.005915231508342109
69	0.999826295402039	0.9999634620928773	0.0038978261576107106	0.9997915508615898	0.5001709144469298	0.006315024477093708
70	0.9999652590804079	0.9999393110650566	0.003806455091140113	0.9998610339077265	0.5002062075531701	0.005716775957233023
71	0.9999652590804079	0.9999441114282299	0.0036942198073219704	0.9997915508615898	0.5001709144469298	0.00562945110262633
72	0.9999305181608157	0.9999290892613968	0.0037298884335704714	0.9997915508615898	0.5001709144469298	0.00555628529719267

epoch	acc	f1'score	loss	val'acc	val'f1'score	val'loss
73	0.9999131477010197	0.999919644623866	0.003703193774590469	0.9997915508615898	0.5001709144469298	0.005516219051713212
74	0.9999652590804079	0.9999540489634245	0.003519717331246061	0.9998610339077265	0.5002062075531701	0.0054023849786678044
75	0.9999652590804079	0.9999496755692909	0.00344312859786618	0.9998610339077265	0.5002062075531701	0.005338032647611168
76	0.9999478886206118	0.9999460576039013	0.003495407624498555	0.9997915508615898	0.5001709144469298	0.00538144701094744
77	0.9999305181608157	0.999931804855327	0.003423143746115526	0.9997915508615898	0.5001709144469298	0.00531559705805694
78	0.9999478886206118	0.9999274413688246	0.003267854001055909	0.9998610339077265	0.5002062075531701	0.005285987166119424
79	0.9999826295402039	0.9999667479447769	0.00322165238745761	0.9998610339077265	0.5002062075531701	0.005108451093337382
80	0.9999478886206118	0.9999435752245598	0.003156957036982487	0.9998610339077265	0.5002062075531701	0.005147117168708327
81	0.9999478886206118	0.999953880022684	0.0031743719732917957	0.999722067815453	0.5001709144469298	0.005684940882255967
82	0.9999826295402039	0.9999841184014668	0.003009517655102647	0.9997915508615898	0.5001709144469298	0.005607723993554157
83	0.9999478886206118	0.9999360993283734	0.002997289070673575	0.9997915508615898	0.5001709144469298	0.0054566330085402395
84	0.9999826295402039	0.9999841184014668	0.002860308930105825	0.9997915508615898	0.5001709144469298	0.005538577215824734
85	0.9999652590804079	0.9999635311358647	0.002794603353910014	0.9997915508615898	0.5001709144469298	0.005322497503069944
86	0.9999826295402039	0.9999831558975997	0.00271688590139455	0.9997915508615898	0.5001709144469298	0.0052592910504203625
87	0.9999652590804079	0.9999524951944025	0.002725089469465581	0.9997915508615898	0.5001709144469298	0.004926927577188573
88	0.9999305181608157	0.9999265040086771	0.002833887700519466	0.9997915508615898	0.5001709144469298	0.005150756155067704
89	0.9999478886206118	0.9999321613849492	0.0026116440472547333	0.9997915508615898	0.5001709144469298	0.004783024852230871
90	0.9999826295402039	0.9999620422746016	0.0025329462455673897	0.9997915508615898	0.5001709144469298	0.004780099682562704
91	0.9999826295402039	0.9999634620928773	0.002480087133982558	0.9997915508615898	0.5001709144469298	0.004757580135106676
92	0.9999826295402039	0.9999667479447769	0.002415170774637595	0.9997915508615898	0.5001709144469298	0.0046344453781495795
93	0.9999478886206118	0.9999362733682623	0.0025883525662166802	0.9997915508615898	0.5001709144469298	0.004534676409077485
94	0.9999826295402039	0.9999646986938322	0.0023074752198459975	0.9997915508615898	0.5001709144469298	0.004717164790944267
95	0.9999478886206118	0.9999269637779243	0.0023038497629507115	0.9997915508615898	0.5001709144469298	0.0047048920407796015
96	0.9999826295402039	0.9999676064156627	0.0022357303075686576	0.9997915508615898	0.5001709144469298	0.004683146017809099
97	0.9999826295402039	0.9999657854409096	0.002186117488903909	0.9997915508615898	0.5001709144469298	0.004420602629442656
98	0.9999305181608157	0.9999325001516632	0.0022242467410325643	0.9997915508615898	0.5001709144469298	0.0047105578290223
99	0.9999305181608157	0.9999060074271586	0.0022827095605323497	0.999722067815453	0.5001356213406897	0.004930893350221743

Table C.6: CNN 5-Fold Cross Validation Fold 1

epoch	acc	f1'score	loss	val'acc	val'f1'score	val'loss
0	0.8664535853251807	0.8717270863344432	0.32484631816038095	0.9483776835961926	0.4724338702510149	0.17274959993805641

epoch	acc	f1'score	loss	val'acc	val'f1'score	val'loss
1	0.9623575597554197	0.9604189448584577	0.12373400669391699	0.96915167709511508	0.4852032445224035	0.0912849129248214
2	0.9822123401889938	0.9813639641868332	0.0736600354609192	0.9868686166886681	0.4946005464817679	0.06177741329001386
3	0.9915404391328516	0.9912339293539292	0.05295182228696146	0.9925658306121031	0.49768484212737146	0.044906191700273886
4	0.9948929961089494	0.9946965911508998	0.04148162182292623	0.99492809000437713	0.49865594867999696	0.03578180711702656
5	0.9964563646470261	0.9962988787720507	0.03448951539102118	0.9958313068852914	0.49891380729349294	0.030310641471590797
6	0.9969948582545859	0.9969368016077056	0.0299655544632197	0.9965260890710762	0.49913992623693787	0.027312628956862862
7	0.9974291272929405	0.9973503872048133	0.0270031633339079265	0.997081914819704	0.49917880365534306	0.0243718314631881
8	0.9976375764313508	0.9975459591225162	0.0251111284903596192	0.9975682623497534	0.4994761980425564	0.02266760136753591
9	0.9979502501389661	0.9978721015581891	0.02372075944344002	0.9977072187869103	0.4995114886966945	0.02172688010114113
10	0.9980023624235687	0.9978738240628457	0.022663539222528586	0.9977072187869103	0.49929161750275575	0.020800599292142747
11	0.998123957754308	0.9980887481409553	0.021848822505824483	0.9981240880983812	0.49954791769063356	0.02027674460336691
12	0.9984018899388549	0.9983525378893586	0.02125530953055751	0.9979851316612242	0.49943391845910917	0.019668267715440564
13	0.9983845191773207	0.998339538808794	0.020762948372484193	0.9982630445355382	0.4996250756610337	0.019360804094099213
14	0.9985234852695942	0.9985113911922936	0.020371501807032216	0.9983325227541165	0.49958320834477166	0.018665878328733888
15	0.998592968315731	0.9985534658095385	0.020066566956317183	0.9981935663169597	0.49954791769063356	0.01851446981067295
16	0.9986971928849361	0.9986266578946795	0.019798375748565588	0.9982630445355382	0.4995897850068956	0.018153718152164513
17	0.9986277098387993	0.9985662955519489	0.01960141309550401	0.998540957409852	0.49958320834477166	0.017965346441534184
18	0.9987145636464703	0.9987536139657857	0.019454282968495086	0.9984714791912735	0.4996654740021276	0.018062117055589449
19	0.9986450806003335	0.9986009347962299	0.019294696021394796	0.998402000972695	0.4996654740021276	0.01768129153229269
20	0.9986971928849361	0.9986123344910152	0.019097134609996142	0.9984714791912735	0.4996654740021276	0.017757102877880213
21	0.9986971928849361	0.9986286044054524	0.019029786849260145	0.9984714791912735	0.4996654740021276	0.01749187522867089
22	0.9987666759310728	0.9987719862112541	0.01885117307376656	0.998540957409852	0.4996654740021276	0.017543373690517446
23	0.9987493051695386	0.9987340936599803	0.01879313783289233	0.9986104356284304	0.4996654740021276	0.017406888551470396
24	0.9987666759310728	0.9987343533493666	0.01872605687193323	0.9986104356284304	0.4996654740021276	0.017182129167740777
25	0.998784046692607	0.9987301949065814	0.018646453223336325	0.9986799138470089	0.4996654740021276	0.017180392793210583
26	0.9988014174541412	0.9987470742197021	0.01856810308727978	0.9986799138470089	0.4996654740021276	0.017221714216600487
27	0.998784046692607	0.9986823066000543	0.018514689422048296	0.9986799138470089	0.4996654740021276	0.01700295677096978
28	0.9987493051695386	0.9987159808719734	0.018515570371401908	0.9986104356284304	0.4996654740021276	0.01703315958403085
29	0.9987493051695386	0.9987589609894638	0.01848068779034103	0.9984714791912735	0.4996654740021276	0.01732585685451145
30	0.9988361589772096	0.9988270518579637	0.01837829839815704	0.9987493920655874	0.4996654740021276	0.016828174876972583
31	0.9988535297387438	0.9988191837135323	0.01836891844397945	0.9986799138470089	0.4996654740021276	0.016870937710992663
32	0.9988361589772096	0.9988174595191387	0.018332984660778165	0.9988883485027444	0.4996654740021276	0.017004203900035538
33	0.998784046692607	0.9987122774654259	0.018399419425904374	0.9986799138470089	0.4996654740021276	0.016744228940466906

epoch	acc	f1'score	loss	val'acc	val'f1'score	val'loss
34	0.9988361589772096	0.9988089881054623	0.0182500497731211	0.9988188702841659	0.4996654740021276	0.01681299231578702
35	0.998784046692607	0.9987088911330587	0.018350332123402282	0.9986799138470089	0.4997007646562657	0.01659288537784725
36	0.9988709005002779	0.9988192981517997	0.01822637964522099	0.9987493920655874	0.4996654740021276	0.016446243132538535
37	0.9988014174541412	0.9987355438187893	0.018253406190527094	0.9987493920655874	0.4996654740021276	0.016460066389470604
38	0.9988535297387438	0.9988732063617887	0.01819644311388149	0.9988188702841659	0.4996654740021276	0.016462887130939956
39	0.9988535297387438	0.9988195239470494	0.018163753863568447	0.9988883485027444	0.4996654740021276	0.0164623265747491
40	0.9988187882156754	0.9987933957424344	0.01817763702466337	0.9987493920655874	0.4996654740021276	0.016398724477819836
41	0.9988535297387438	0.9988155243064113	0.018140204513101468	0.9987493920655874	0.4996654740021276	0.016394662489507033
42	0.9988361589772096	0.998817924229949	0.01815126105960576	0.9987493920655874	0.4996654740021276	0.01712859046668563
43	0.9988187882156754	0.9987477720148236	0.018174074633986497	0.9987493920655874	0.4996654740021276	0.016191897572659777
44	0.9988535297387438	0.9987897714222054	0.018124340498002055	0.9988188702841659	0.4996654740021276	0.016181221113055082
45	0.9988709005002779	0.9988279683911979	0.018081758390290435	0.998540957409852	0.4997007646562657	0.016878145223120988
46	0.9988535297387438	0.9988571500168567	0.018097214258107414	0.9988883485027444	0.4996654740021276	0.01612333678647742
47	0.9988709005002779	0.9988496810810774	0.018081180841539767	0.9988883485027444	0.4996654740021276	0.016294168724864354
48	0.9988709005002779	0.9988574517177435	0.018050565558907876	0.9988883485027444	0.4996654740021276	0.016156740348788702
49	0.9988535297387438	0.9988314435175885	0.018096437653532408	0.9988188702841659	0.4996654740021276	0.0162461358224753

Table C.7: CNN 5-Fold Cross Validation Fold 2

epoch	acc	f1'score	loss	val'acc	val'f1'score	val'loss
0	0.8562594451875141	0.86346060609364113	0.31767662768626137	0.9382295719844358	0.46185962355800611	0.1602621654147182
1	0.9623929545415068	0.9605358129513121	0.11497938462958962	0.9833240689271817	0.49587734319422894	0.08276268317179697
2	0.987979641821119	0.9875722386570752	0.06443569391186305	0.9941634241245136	0.49814204363905107	0.04470650085059808
3	0.9948062325209748	0.9946598689832645	0.036311112004425224	0.9972206781545303	0.49934369978340154	0.02525668967092707
4	0.997793951605899	0.997695893596428	0.021863494175737058	0.9986103390772652	0.4997873970241663	0.015147134658847511
5	0.9986277336761104	0.9985348662497814	0.013973812675724923	0.9991662034463591	0.5001213594301996	0.010247251544420027
6	0.9990793656308082	0.99903644481796122	0.009428881492912072	0.9995136186770428	0.5000860663239594	0.006475469425838465
7	0.99932252520679533	0.99929444497778362	0.006801953902204634	0.9996525847693163	0.5001213594301996	0.004529740829440812
8	0.9994962566659139	0.9994864267056676	0.005060620299962305	0.9996525847693163	0.5001213594301996	0.00325136288296643
9	0.9996178498844864	0.9995883293001439	0.003990234993391301	0.9997915508615898	0.5001985227617706	0.0025241896044718476
10	0.9996873317236707	0.9996695441048717	0.0032623417196527738	0.9997915508615898	0.5001985227617706	0.0019265721406987813
11	0.9997915544824472	0.9997530968986591	0.0027325832857477412	0.999722067815453	0.5001632296555303	0.0015825426595534044

epoch	acc	f1'score	loss	val'acc	val'f1'score	val'loss
12	0.9998089249422433	0.9997915264863104	0.0023200557613277644	0.9998610339077265	0.5002389239098642	0.0011795930581549644
13	0.9998089249422433	0.9997787032286163	0.002047827851908054	0.9995831017231795	0.5002389239098642	0.002080679840644295
14	0.9998089249422433	0.9997990591036637	0.0018297675013725005	0.9998610339077265	0.5002389239098642	0.0008225999793584216
15	0.9998436658618354	0.9998171192623687	0.001632635745375474	0.9999305169538633	0.5002389239098642	0.0007333087144958927
16	0.9998436658618354	0.9998248973843056	0.0015151515745575122	0.9999305169538633	0.5002389239098642	0.0005811521341500287
17	0.9998262954020393	0.9997967609024572	0.001387550911133777	0.9998610339077265	0.5002389239098642	0.000534644447167734
18	0.9998784067814275	0.9998494391621995	0.0012991662849600133	0.9999305169538633	0.5002389239098642	0.00047438516609724936
19	0.9998610363216315	0.9998651776926052	0.0012500520615049736	0.9999305169538633	0.5002389239098642	0.00044080376170105014
20	0.9998957772412236	0.9999035708987385	0.001164861218838187	0.9999305169538633	0.5002389239098642	0.0003382239701125387
21	0.9998957772412236	0.9998876759544139	0.001112818669129937	0.9999305169538633	0.5002389239098642	0.00034069478425371845
22	0.9998784067814275	0.9998691830617843	0.0011146647927772764	0.9999305169538633	0.5002389239098642	0.00030236102095257823
23	0.9998957772412236	0.9998776524760492	0.0010505215898981499	0.9999305169538633	0.5002389239098642	0.00028423649299889235
24	0.9999131477010197	0.9999173661027838	0.0010120284390307515	0.9998610339077265	0.5002389239098642	0.0002906595663625144
25	0.9999131477010197	0.9998837614318702	0.0010181914759480135	0.9999305169538633	0.5002389239098642	0.0001912539461360108
26	0.9999305181608157	0.9999083474734785	0.0009698628791528636	0.9999305169538633	0.5002389239098642	0.00020754873003699613
27	0.9999131477010197	0.9998924015035028	0.0009664377017210712	0.9999305169538633	0.5002389239098642	0.0001424501364902852
28	0.9999131477010197	0.9999157823166022	0.0009475553222890022	1.0	0.500277932184547	0.00013804061459592412
29	0.9999131477010197	0.9999043361375193	0.0009313564729126418	0.9999305169538633	0.500277932184547	0.00021603474889452472
30	0.9999305181608157	0.9999307603822725	0.000926476956439052	0.9999305169538633	0.5002389239098642	0.00012566255811737942
31	0.9999478886206118	0.999925127198224	0.0009128390440350547	0.9998610339077265	0.500277932184547	0.00040643812336699356
32	0.9999478886206118	0.9999509723008535	0.0009011280264127307	0.9998610339077265	0.500277932184547	0.00041182586454810893
33	0.9999478886206118	0.9999330597461884	0.0008943549586969659	1.0	0.500277932184547	0.00010202912402329144
34	0.9999478886206118	0.9999384099570071	0.0008925097346345223	1.0	0.500277932184547	0.00010168379453127969
35	0.9999478886206118	0.9999464587603787	0.0008820569002250551	1.0	0.500277932184547	0.00010984568604410926
36	0.9999305181608157	0.9999335868987407	0.000917170108175327	1.0	0.500277932184547	0.00011184783532536387
37	0.9999478886206118	0.9999255553237609	0.0008726132726724116	1.0	0.500277932184547	6.546162754021648e-05
38	0.9999478886206118	0.9999566715522643	0.000871893638487755	1.0	0.500277932184547	6.391390598205303e-05
39	0.9999478886206118	0.9999325573729091	0.0008689489007358759	1.0	0.500277932184547	6.586828163926283e-05
40	0.9999478886206118	0.9999493434558009	0.0008678540836141178	1.0	0.500277932184547	7.49078777863812e-05
41	0.9999478886206118	0.9999493434558009	0.000863788643305505	0.9999305169538633	0.5002426390783067	0.00012123465254840974
42	0.9999478886206118	0.9999509723008535	0.0008580396756768066	1.0	0.500277932184547	6.717751737281296e-05
43	0.9999478886206118	0.9999260258907785	0.0008570285061514593	0.9999305169538633	0.5002426390783067	0.00012190454386088859
44	0.9999478886206118	0.9999493442509574	0.0008550595619331394	1.0	0.500277932184547	4.145689163879047e-05

epoch	acc	f1`score	loss	val`acc	val`f1`score	val`loss
45	0.9999478886206118	0.9999471443509788	0.0008525845422524721	1.0	0.500277932184547	6.47928087285737e-05
46	0.9999478886206118	0.999945698027485	0.0008535100974405419	1.0	0.500277932184547	4.4586732924084e-05
47	0.9999478886206118	0.999948106854846	0.0008501929459978677	1.0	0.500277932184547	6.677540587120655e-05
48	0.9999478886206118	0.9999292715210094	0.0008488594464791725	0.9999305169538633	0.5002426390783067	0.00012308633693415444
49	0.9999478886206118	0.9999532606619979	0.0008479404167568994	1.0	0.500277932184547	3.356211934396431e-05

Table C.8: CNN 5-Fold Cross Validation Fold 3

epoch	acc	f1`score	loss	val`acc	val`f1`score	val`loss
0	0.8732651253278674	0.8769709089546596	0.2912143414380046	0.9517787659811006	0.4775638188700864	0.1422898507910108
1	0.9735274192707881	0.9723528058046547	0.10669492119687528	0.9863813229571985	0.4952763218956566	0.07197103303625602
2	0.9894908718223771	0.9891548719642853	0.05999761952913935	0.9939549749861034	0.4985383175822879	0.040377038589557296
3	0.994840973440567	0.9946235805202706	0.035177797813091896	0.9979849916620345	0.49981998191269456	0.023505745059794317
4	0.9974639128697737	0.99736834585384	0.022187450277820094	0.9987493051695386	0.49989056812517507	0.014930006999291012
5	0.9982629540203929	0.998212829439029	0.014978960072892409	0.9988882712618121	0.4999269996053116	0.010809158973904333
6	0.9986103632163144	0.9985310383661697	0.010422100672889402	0.9991662034463591	0.5000779384901419	0.007134751155362243
7	0.9989404019524397	0.9989056553342256	0.0075598758385028815	0.9991662034463591	0.5000779384901419	0.005335455476933185
8	0.9991314770101964	0.9991146898940153	0.005891355343703213	0.9995136186770428	0.5001213594301996	0.003951387779844997
9	0.9993051816081572	0.999299460718756	0.0045968322976230715	0.9995831017231795	0.5001213594301996	0.003102709034888845
10	0.9993746634473415	0.999376658061299	0.0037988761349707595	0.9995831017231795	0.5001213594301996	0.0024932325095074419
11	0.9994788862061179	0.999447761194655	0.0032019566305223133	0.9995831017231795	0.5001213594301996	0.0019541499946423182
12	0.9995657385050982	0.9995333848742712	0.002732192892884938	0.9996525847693163	0.5001566525364399	0.0016441716180229475
13	0.9997220726432628	0.9997118368881259	0.0023736888596859814	0.999722067815453	0.5001213594301996	0.0013491070964920429
14	0.9997741840226511	0.999717691628879	0.0021059540111183503	0.9996525847693163	0.5000860663239594	0.0012138826943633248
15	0.9998610363216315	0.9998407040362282	0.0018606561833975769	0.9997915508615898	0.5001566525364399	0.0010734856020990337
16	0.9998262954020393	0.9998098505676195	0.0017118288971516378	0.999722067815453	0.5001213594301996	0.0009411976149232193
17	0.9998610363216315	0.9998483674237018	0.0015670910266935767	0.9998610339077265	0.5001985227617706	0.0006840753936630578
18	0.9998436658618354	0.9998426336461382	0.0014248192160620594	0.9998610339077265	0.5001985227617706	0.0006383558424155705
19	0.9998784067814275	0.9998502997203712	0.0013102794261819653	0.9998610339077265	0.5001985227617706	0.0005609623406244874
20	0.9998957772412236	0.9998788976880939	0.0012547630507104847	0.9998610339077265	0.5001985227617706	0.000505178148209902
21	0.9998957772412236	0.9998946367848416	0.0012013855645069654	0.9998610339077265	0.5001985227617706	0.00044599059897733876
22	0.9999131477010197	0.9998924658780515	0.0011435629446814582	0.9998610339077265	0.5001985227617706	0.0004985024298475092

epoch	acc	fl'score	loss	val'acc	val'fl'score	val'loss
23	0.9999131477010197	0.9999063028909724	0.0010946747671049836	0.9998610339077265	0.5001985227617706	0.0004111797759827371
24	0.9999305181608157	0.9999339422011906	0.0010608801952883236	0.9998610339077265	0.5001985227617706	0.00032682876758468373
25	0.9999305181608157	0.999201328530911	0.001029295125081512	0.9998610339077265	0.5001985227617706	0.0002666499079030257
26	0.9999305181608157	0.9999183046225346	0.0010024044666744254	0.9999305169538633	0.5002389239098642	0.00024232058998592028
27	0.9999305181608157	0.9999065876595165	0.0009996076561836196	0.9998610339077265	0.5001985227617706	0.0002362208754435793
28	0.9999305181608157	0.9999164998791217	0.0009518024800431894	0.9999305169538633	0.5002389239098642	0.0002390313642827984
29	0.9999305181608157	0.9999348234996955	0.0009310486149180815	0.9999305169538633	0.5002389239098642	0.00021063264123155604
30	0.9999305181608157	0.9999316639120291	0.0009453650467603286	0.9999305169538633	0.5002389239098642	0.0002048642424859825
31	0.9999478886206118	0.9999334158437948	0.0009418685610983977	0.9999305169538633	0.5002389239098642	0.00020213004563563158
32	0.9999305181608157	0.9999104822037519	0.0009089788832379796	0.9999305169538633	0.5002389239098642	0.00022575148594711705
33	0.9999305181608157	0.9999359491763123	0.0009154396820458224	0.9999305169538633	0.5002389239098642	0.00016820442478793083
34	0.9999478886206118	0.9999396580183627	0.000897722328044423	0.9999305169538633	0.5002389239098642	0.00015621394265914341
35	0.9999478886206118	0.9999178253419266	0.0008893191949162904	0.9999305169538633	0.5002389239098642	0.00013168167800517027
36	0.9999478886206118	0.9999504309980349	0.0008789164341098574	0.9999305169538633	0.5002389239098642	0.0001056971720430179
37	0.9999478886206118	0.999947251796507	0.0008765821044151691	0.9999305169538633	0.5002389239098642	0.00019717906355271576
38	0.9999478886206118	0.9999504309980349	0.0008685753548186479	0.9999305169538633	0.5002389239098642	0.00011703204334632924
39	0.9999478886206118	0.999942716753674	0.0008664963619716313	0.9999305169538633	0.5002389239098642	0.00010050266920151006
40	0.9999478886206118	0.9999263637991789	0.0008653612287483779	0.9999305169538633	0.5002389239098642	0.0001019926077957696
41	0.9999478886206118	0.9999472291345455	0.0008667212265493576	0.9999305169538633	0.5002389239098642	8.941659778737953e-05
42	0.9999478886206118	0.9999546793538018	0.0008630393724813044	0.9999305169538633	0.5002389239098642	9.952359817868623e-05
43	0.9999478886206118	0.9999539840556655	0.0008554471195089333	0.9999305169538633	0.5002389239098642	0.00011139407857775811
44	0.9999478886206118	0.9999403344315311	0.0008538263682811022	0.9999305169538633	0.5002389239098642	7.390763758038783e-05
45	0.9999478886206118	0.9999333130667052	0.0008516737709932275	0.9999305169538633	0.5002389239098642	9.907575788195784e-05
46	0.9999478886206118	0.9999439533546288	0.0008526544655117107	0.9999305169538633	0.5002389239098642	7.447625875758666e-05
47	0.9999478886206118	0.9999438023411481	0.0008498383915756777	0.9999305169538633	0.5002389239098642	9.007096655849654e-05
48	0.9999478886206118	0.9999435510385482	0.0008486207245448404	0.9999305169538633	0.5002389239098642	7.651657331257082e-05
49	0.9999478886206118	0.9999192208085331	0.0008474853947901493	0.9999305169538633	0.5002389239098642	6.360705959602189e-05

Table C.9: CNN 5-Fold Cross Validation Fold 4

epoch	acc	fl'score	loss	val'acc	val'fl'score	val'loss
0	0.8553214403585263	0.8616138776382906	0.3231440335010012	0.9469149527515286	0.4699937189334363	0.16417919685927584

epoch	acc	f1'score	loss	val'acc	val'f1'score	val'loss
1	0.9564001459118623	0.9542257312606499	0.12226574722368554	0.9671345191773207	0.484858298420523	0.0936996879411183
2	0.9775747364032726	0.9766595724098265	0.07602631551172132	0.9846442468037799	0.4919842312639458	0.061605957483845016
3	0.9900640969966475	0.9897573998790301	0.05137298272297637	0.9931211784324625	0.4975092104900672	0.04153435469614085
4	0.9938855981517831	0.99356123607137	0.03599826463649334	0.9952056698165648	0.49866815432367756	0.029771956863717003
5	0.9959005714881273	0.9957762460454003	0.02642225568381239	0.9968732629238466	0.49886142644304377	0.022899721527642282
6	0.9972902082718129	0.9971745788267289	0.020179658981831234	0.9976375764313508	0.4992705413273403	0.017729058783972438
7	0.9977592106863069	0.9976534708019725	0.016053922631661974	0.9977765425236242	0.49926820856256576	0.014500934852325653
8	0.998245835605969	0.9980970736303172	0.01316456609106106	0.9977765425236242	0.49967175129057634	0.013136996968919326
9	0.9986277336761104	0.9985561747555023	0.011178637224169015	0.9984713729849917	0.4996753383040627	0.010399801962619236
10	0.9989056610328475	0.9988360667784066	0.009541338310566064	0.99854085660311284	0.49960475209158217	0.009144351320855735
11	0.9991314770101964	0.9991172340998843	0.008352845414788371	0.9988187882156754	0.4997459245165432	0.008045241791298086
12	0.9992356997689729	0.9992167747087606	0.007298822353779027	0.9986103390772652	0.49956945898534194	0.008261100096842859
13	0.9993051816081572	0.9992729840961715	0.006586728710978045	0.9990272373540856	0.4998917065440714	0.006822172451304957
14	0.9994267748267297	0.9994009102034211	0.005932012080077456	0.9990967204002223	0.49981998191269456	0.006299951843145852
15	0.9994615157463218	0.9993985061138678	0.005509831278346559	0.9992356864924958	0.5000073522776614	0.00594457805122623
16	0.9994615157463218	0.9994215183118573	0.005046369457216608	0.9992356864924958	0.4998917065440714	0.005453538662489242
17	0.9995483680453021	0.9994983441806613	0.00464038731399943	0.9991662034463591	0.4998917065440714	0.005359095268774247
18	0.9995309975855061	0.9995200626558447	0.004437923615642047	0.9992356864924958	0.4998917065440714	0.0051852301069204385
19	0.9995657385050982	0.9995460082194633	0.004154968076116994	0.9995136186770428	0.5000507732177192	0.004687125528979671
20	0.9995831089648943	0.9995519281899725	0.0038688816687531815	0.9993746525847693	0.49998018700523866	0.00452420165701793
21	0.9996699612638746	0.9996745056798226	0.003725570701748384	0.9994441356309061	0.5000507732177192	0.004334318717910791
22	0.9996352203442825	0.9996333125318084	0.0035460109702063845	0.9995831017231795	0.5000860663239594	0.004019616993743548
23	0.9996873317236707	0.9996578115338003	0.0033825725672937726	0.9993051695386326	0.5000860663239594	0.004479003611893658
24	0.9996873317236707	0.9996349589365681	0.0032678924570497016	0.9993746525847693	0.49997904858634235	0.0041400181220606685
25	0.9996873317236707	0.9996833314205086	0.0031307430855920843	0.9992356864924958	0.49987316926762154	0.004594801079494208
26	0.9996699612638746	0.9996379778809205	0.0030892985676392584	0.9995136186770428	0.5000507732177192	0.00385902377742093
27	0.9997047021834667	0.9996707871996051	0.002948681194991306	0.9993746525847693	0.4999448938989984	0.0036843822254102763
28	0.999739443103059	0.9997169194956353	0.0028578949311967214	0.9993746525847693	0.5000154801114789	0.003650887949981024
29	0.9996873317236707	0.9996809586402443	0.002854530178840309	0.9974291272929405	0.4989463293002406	0.009282956904016204
30	0.9997220726432628	0.9996817850760537	0.002801033560122058	0.9992356864924958	0.5001279365492901	0.004415752652348846
31	0.999739443103059	0.9997149300919949	0.0026990426755218546	0.9995136186770428	0.5000507732177192	0.0034207592614365124
32	0.9997741840226511	0.9997574394173196	0.0026550983438856962	0.9993051695386326	0.5001279365492901	0.004108609677799448
33	0.9997741840226511	0.9997530324609535	0.002622239083312147	0.9993746525847693	0.5000154801114789	0.0033594303693540105

epoch	acc	f1'score	loss	val'acc	val'f1'score	val'loss
34	0.9998089249422433	0.9997938251182269	0.00252295905020197	0.9994441356309061	0.5000154801114789	0.0032353742373190083
35	0.9997915544824472	0.9997660391347575	0.0024926111072218022	0.9995136186770428	0.5000154801114789	0.003178452976144281
36	0.9997915544824472	0.9997957371352982	0.0024647541987653796	0.9995831017231795	0.5001279365492901	0.0030078872053060277
37	0.9998262954020393	0.9998124029870001	0.002445018369200969	0.9994441356309061	0.5000154801114789	0.0031540230025477356
38	0.9998089249422433	0.9997857741582002	0.002396601791858147	0.9994441356309061	0.5000154801114789	0.0031758214538717723
39	0.9998089249422433	0.9998148622736703	0.002355796597692748	0.999722067815453	0.5001330445911435	0.00282888116184562754
40	0.9998089249422433	0.9998139406541022	0.0023733693741740773	0.9990272373540856	0.4997649571841263	0.004618656996334742
41	0.9998262954020393	0.9998261001548697	0.0022919137341958938	0.9995136186770428	0.5000573503368095	0.003044594034639219
42	0.9997741840226511	0.9997473697173747	0.002348996676449693	0.9996525847693163	0.5000977514849032	0.002869891164201633
43	0.9997915544824472	0.9997880151712925	0.0023085640047183194	0.9995136186770428	0.5000573503368095	0.0029235651074644878
44	0.9998089249422433	0.9998037291212144	0.0022451650779597354	0.9995831017231795	0.5001330445911435	0.0028126507709762267
45	0.9997741840226511	0.999758198987504	0.0023073384830818644	0.9996525847693163	0.5000977514849032	0.0029264937027651777
46	0.9998089249422433	0.999787503723082	0.002207815863505789	0.9996525847693163	0.5000977514849032	0.0030114640399768306
47	0.9998436658618354	0.9998401979522183	0.0021982418344321207	0.9996525847693163	0.5000977514849032	0.002887423095004375
48	0.9998262954020393	0.9997929164530841	0.0022127324430852046	0.9996525847693163	0.5000977514849032	0.003115709879788671
49	0.9998436658618354	0.9998430635939088	0.002188452986503421	0.9995136186770428	0.5000977514849032	0.002988445172830406

Table C.10: CNN 5-Fold Cross Validation Fold 5

epoch	acc	f1'score	loss	val'acc	val'f1'score	val'loss
0	0.8799527523493547	0.8843476052039398	0.2950531428201908	0.950319622012229	0.4700184477044848	0.1605703019059057
1	0.968837395125849	0.9676594031027093	0.11647189679090832	0.9801973318510283	0.49080571563724945	0.08919700695769088
2	0.9887786829717382	0.9884154947555842	0.06680259444084187	0.9918704836020011	0.49691717673699815	0.05912629992879571
3	0.9933992252774931	0.9930627874828275	0.045013936410085616	0.9945108393551974	0.4986003370269131	0.04409975785094999
4	0.995813719189147	0.9955083635337539	0.034325032407370906	0.9963173985547527	0.49870477019580883	0.03740903150121341
5	0.9970817627542601	0.9968757862971767	0.028834400261610518	0.9965953307392996	0.49912270126639635	0.03271405383418765
6	0.9975855060883462	0.9974819674623828	0.025570718611847905	0.9970817120622568	0.49904282759665913	0.030093816152559526
7	0.9978981743646754	0.997811250001527	0.023547028128879936	0.9974291272929405	0.4992297190040135	0.02796275725255818
8	0.9980718789626362	0.9980872019243521	0.02198928862416514	0.9974291272929405	0.4992650121102537	0.02548085792044912
9	0.9981934721812087	0.9980985898281926	0.02096194028439961	0.9977765425236242	0.4991851384405165	0.02514775217478265
10	0.9983150653997811	0.9982046252788412	0.02020870994975034	0.998193440804447	0.499500299891089903	0.024041714343128972
11	0.9984887699977418	0.9983645917583703	0.019562947440722813	0.998123957754308	0.4993801576816883	0.023027341853254873

epoch	acc	f1'score	loss	val'acc	val'f1'score	val'loss
12	0.9986277336761104	0.9985674963600596	0.019040677130440435	0.9981934408004447	0.49950029891089903	0.023035804894719035
13	0.9987145859750908	0.9986240378216169	0.018604303113382625	0.9982629238465814	0.49950029891089903	0.02287490968219218
14	0.998766697354479	0.9986705518290268	0.018287937266320366	0.9982629238465814	0.49950029891089903	0.02242715362848598
15	0.9988361791936633	0.9987858643017267	0.01803944589318639	0.99854085660311284	0.49960731664851615	0.021600929054051267
16	0.9988709201132554	0.9988149012321003	0.01779074302448897	0.9984713729849917	0.49957202354227587	0.021374589789830853
17	0.9988535496534593	0.998835718532971	0.017640181283771695	0.99854085660311284	0.49957202354227587	0.0212503440871182
18	0.9988535496534593	0.9988185811503314	0.017500987200490502	0.9984713729849917	0.49957202354227587	0.02123038324913329
19	0.9988882905730515	0.9988865953624527	0.017306074804801768	0.9984713729849917	0.49957202354227587	0.02102760808099103
20	0.9989056610328475	0.9988462249695428	0.017210965193367338	0.9986103390772652	0.49957202354227587	0.020945894919306964
21	0.9989230314926436	0.9988608636027625	0.017120705860092598	0.9984713729849917	0.49957202354227587	0.020904407907557487
22	0.9989404019524397	0.9988873808808315	0.01704130613475675	0.9986103390772652	0.49957202354227587	0.020946601961553818
23	0.9989056610328475	0.9988809036318975	0.017005490553667825	0.9986798221234019	0.49960731664851615	0.020699251533222694
24	0.9989925133318279	0.998913961768318	0.01660273076899437	0.9986798221234019	0.49960731664851615	0.020676489609885558
25	0.9989925133318279	0.9989235612656923	0.016574616351438712	0.9986798221234019	0.49960731664851615	0.02063663372875479
26	0.9989925133318279	0.9989770432268072	0.016522438238746672	0.9986798221234019	0.49960731664851615	0.02061160038349703
27	0.9989925133318279	0.9989357073813245	0.016498438674225253	0.9986103390772652	0.49957202354227587	0.02057101358958163
28	0.9989925133318279	0.9989320664920273	0.016440674480071707	0.9986103390772652	0.49960731664851615	0.020669131878781447
29	0.9989925133318279	0.9989504055487847	0.01640867569119207	0.9986103390772652	0.49957202354227587	0.02052189664287039
30	0.9989925133318279	0.9989887732105139	0.016400459552801634	0.99854085660311284	0.49953673043603564	0.020670843477179582
31	0.9989925133318279	0.9989200465081021	0.0163697697191589	0.9986103390772652	0.49957202354227587	0.020459904264539377
32	0.9989925133318279	0.998966976677463	0.01634095192756452	0.9986103390772652	0.49957202354227587	0.020471631853790046
33	0.9989925133318279	0.9989321120147395	0.016329989271885104	0.9986103390772652	0.49957202354227587	0.020422143393491188
34	0.9989925133318279	0.9989418962804039	0.016313906572810122	0.9986798221234019	0.49960731664851615	0.020367256564822998
35	0.9989925133318279	0.9989380299010688	0.01629526791777656	0.9986798221234019	0.49957202354227587	0.020474751920215452
36	0.9989751428720318	0.9989735832687556	0.016331582834327812	0.9986103390772652	0.49953673043603564	0.020629387382620445
37	0.9989925133318279	0.998937176429937	0.016284020641617106	0.99854085660311284	0.49960731664851615	0.020393526918418958
38	0.9989925133318279	0.998979629244658	0.0162820143131664	0.9986798221234019	0.49957202354227587	0.02037394496949966
39	0.9989751428720318	0.9989440784881494	0.016294954859438212	0.9986103390772652	0.49960731664851615	0.020377683217814926
40	0.9989925133318279	0.9989076660169718	0.0162707041811131013	0.9986798221234019	0.49960731664851615	0.020373128232486333
41	0.9989925133318279	0.998940845315371	0.016269202045892123	0.9986798221234019	0.49957202354227587	0.020467368708193927
42	0.9989925133318279	0.9988954222327165	0.016265588553212237	0.9986103390772652	0.49957202354227587	0.020474026267725792
43	0.9989925133318279	0.9989724426829537	0.016260463454080297	0.9986103390772652	0.49957202354227587	0.02046953985957601
44	0.9989925133318279	0.9989764206523634	0.01626377494848257	0.9986103390772652	0.49957202354227587	

epoch	acc	f1`score	loss	val`acc	val`f1`score	val`loss
45	0.998925133318279	0.9989339311341272	0.01625546672740062	0.9986798221234019	0.49960731664851615	0.020423946524762007
46	0.998925133318279	0.9989387776132741	0.01627990515402744	0.9986103390772652	0.49957202354227587	0.020794976486460528
47	0.998925133318279	0.9989472592168334	0.016252493522597982	0.9986103390772652	0.49957202354227587	0.02056960535208382
48	0.998925133318279	0.998957458189768	0.01625090873493161	0.9986103390772652	0.49957202354227587	0.020610088047345262
49	0.998925133318279	0.9989840626399794	0.01625042681944547	0.9986103390772652	0.49957202354227587	0.020601900655046506

Table C.11: LSTM 5-Fold Cross Validation Fold 1

epoch	acc	f1`score	loss	val`acc	val`f1`score	val`loss
0	0.569239855475264	0.48727670356241976	0.685723853614875	0.5658306121031057	0.19510113971821255	0.6442687899311182
1	0.829992356864925	0.8097203093777106	0.38450192914382164	0.938650732995206	0.4455603011527585	0.1657107103100158
2	0.8351688438021123	0.812230003980082	0.3736104141794489	0.8304731466229659	0.43131830077236594	0.38077922817326104
3	0.8383476931628683	0.8259200374241203	0.37905756523703255	0.8410338358924477	0.3480126871175136	0.31247843341678144
4	0.6636673151750972	0.6423721107891893	0.5937606559785755	0.6902661015895792	0.3837286264629864	0.5672091930237073
5	0.6700423846581434	0.6482693676470782	0.5786739534441135	0.6926283610005413	0.2885445676485675	0.559486643425131
6	0.6744371873262924	0.655637793959413	0.5732059836752087	0.7005488779764647	0.37655808945009595	0.5490150736452167
7	0.6808122568093385	0.6612872875121119	0.5687811835697454	0.6858194956171217	0.3227896631894562	0.55737672232356612
8	0.6831746803779878	0.6635618925160869	0.5676479622382068	0.7129854790357522	0.3626531461177538	0.5427284869484778
9	0.6833136464702613	0.6632410803723826	0.5655571297680821	0.7252831237324249	0.3457143744708085	0.54111348577620114
10	0.6840953307392996	0.6652190925479929	0.5651460596095728	0.7118043493861778	0.3759101411842914	0.5352776551034613
11	0.6838000277932185	0.6632485899115483	0.5636304805946721	0.723476690078373	0.3623794802679824	0.5346493917426658
12	0.6878474152306837	0.6688178081340826	0.5617908047198454	0.7213228653024403	0.36352894187412543	0.5328550458757805
13	0.6914605336297943	0.6715715810151283	0.5616990926697759	0.7165983464308215	0.383722212683506	0.5338013316015798
14	0.6877084491384102	0.6681119354981194	0.5618204296198469	0.7247967762023755	0.36018585080377574	0.5337081724655005

Table C.12: LSTM 5-Fold Cross Validation Fold 2

epoch	acc	f1`score	loss	val`acc	val`f1`score	val`loss
0	0.686689016658271	0.6347955635726689	0.5784452570262271	0.8447053918843802	0.34621465703127713	0.345698080775227
1	0.8643540794524831	0.8379127412822708	0.31597692374760655	0.959769316286826	0.4714330283551166	0.14393599711414456
2	0.8635550383018639	0.8431595835504915	0.3249835001345869	0.8428988326848249	0.3464396011836003	0.4599894106842623

epoch	acc	f1'score	loss	val'acc	val'f1'score	val'loss
3	0.9603258698257743	0.9588651668534868	0.12106777150378936	1.0	0.500277932184547	0.0007196214878227585
4	0.9933123729785127	0.9928047040393887	0.03206281919949729	1.0	0.500277932184547	4.882598684276837e-05
5	0.9925133318278935	0.991722105174517	0.032934770154118005	0.8597832128960534	0.36185498518238735	0.32762924259517645
6	0.9600826833886293	0.9567844211743562	0.11359802068547886	0.9998610339077265	0.5002073459720665	0.0005016455383957226
7	0.9602216470669979	0.9573153600105244	0.11290933726427023	0.8156614785992218	0.4955695775431219	0.3582594286141272
8	0.9922527749309524	0.9914211087557437	0.031674089890076786	1.0	0.500277932184547	2.2920251732856552e-05
9	0.9371363059980198	0.9333821155909843	0.19217811096373041	0.7907170650361312	0.4978388805665063	2.0909061262384316
10	0.9367541558825062	0.933801040271052	0.17904574639064597	0.8187882156753752	0.39974241738123256	0.3855692854533184
11	0.9227014539074849	0.9192714937329324	0.2036072708401402	0.9861033907726515	0.4937347708442862	0.04373199085534933
12	0.9893866490646007	0.9888593945055296	0.04208305859046843	1.0	0.500277932184547	0.0012522940453387312
13	0.9985235109173339	0.998491871072033	0.011741216812060475	0.9999305169538633	0.500277932184547	0.0004882260162765502
14	0.9754902812277441	0.9739788519642253	0.07012062282529102	0.9931906614785992	0.4973919925721504	0.02473620668043532

Table C.13: LSTM 5-Fold Cross Validation Fold 3

epoch	acc	f1'score	loss	val'acc	val'f1'score	val'loss
0	0.6249370320832393	0.5519962062594496	0.6400112977958723	0.6350055586436909	0.1662133895171358	0.6304205202514626
1	0.77135263770432	0.721883137591504	0.47454642201572006	0.8491523068371317	0.34979366971228504	0.39831829755119214
2	0.8509267140301203	0.8233332120019421	0.3367426961253165	0.9175931072818232	0.4486551782434155	0.20470369454957815
3	0.922249821952787	0.9118498339652436	0.18651282085595566	0.9455252918287937	0.4568508609988545	0.12629889297542918
4	0.9354166304782088	0.9278033202493248	0.17724656858961696	0.9968732629238466	0.49852113845681006	0.011768433075913826
5	0.9589709739616807	0.9558484724597152	0.1100790680986548	0.9989577543079489	0.500277932184547	0.006846805592598171
6	0.9688721360454411	0.9668193992638419	0.10092877827464633	0.9988882712618121	0.49997671582156783	0.005453833565478311
7	0.9929302228629991	0.9926008061380818	0.029213586491011325	1.0	0.500277932184547	0.00043241565996856576
8	0.9970296513748719	0.996687326376773	0.015154852855435816	0.9996525847693163	0.5001003282344494	0.0036148715172557478
9	0.9810488283624867	0.9793569324452209	0.06109590798429385	0.9742217898832685	0.500297420219688	0.07948936978522976
10	0.9924612204485053	0.9919792435940729	0.029760901195466906	0.9999305169538633	0.5002389239098642	0.00016951665404537307
11	0.9900293560770553	0.9893192134779584	0.04126329707227568	0.9996525847693163	0.500168376973838	0.0030565383849104834
12	0.997047021834668	0.9968922761210982	0.017829827241015166	0.9999305169538633	0.5002389239098642	0.0007448765157411064
13	0.9912626587225764	0.9906968683630363	0.04011419434361847	1.0	0.500277932184547	2.4337723704576587e-05
14	0.9519880491236603	0.9489210701564016	0.1320958820530095	0.9995831017231795	0.5000213792153104	0.0025938209380677282

Table C.14: LSTM 5-Fold Cross Validation Fold 4

epoch	acc	f1_score	loss	val_acc	val_f1_score	val_loss
0	0.7310705414372318	0.657983763602376	0.5090355724505266	0.9262090050027794	0.461509497315437	0.21637794935145002
1	0.8433184526394414	0.82496448056260289	0.36039066326523017	0.8321984435797666	0.3451836236388634	0.39470708749985417
2	0.8086817558060762	0.788159831978642	0.3818459727968302	0.9695664257921067	0.4732587782789297	0.09104470905949832
3	0.8618353627820529	0.8462672272438607	0.2961866131821809	0.9164813785436353	0.43070647870917794	0.291649356689418
4	0.8784762632666887	0.8678185124463458	0.2972252629070283	0.9305169538632574	0.45032673232617676	0.1615102588788727
5	0.9336448435790096	0.9283299719441019	0.1708090406689798	0.975958866036687	0.479701339237686	0.06578592792062245
6	0.9657107123625562	0.962601238407254	0.10590914145029444	0.9178015564202334	0.4394461091125323	0.20972021368475607
7	0.9696711771960603	0.9677755978243747	0.09135908994358236	0.9232907170650362	0.43142276753843856	0.3676153789775259
8	0.9389428338168111	0.9336583196239902	0.1456135480102929	0.9998610339077265	0.500277932184547	0.0011177259399501266
9	0.9679167607566572	0.9643439900575044	0.08831043661728287	0.9993746525847693	0.4999097773205775	0.0032038780943562523
10	0.9929475933227953	0.9926939527803187	0.02941384244405657	1.0	0.500277932184547	0.0006950269670940026
11	0.9982108426410047	0.997906579235628	0.013023436075569243	1.0	0.500277932184547	4.966154886280089e-05
12	0.9747086105369209	0.9732551088636532	0.07853448873482198	0.9997915508615898	0.5002426390783067	0.0011369560192358017
13	0.9818131285935139	0.9807426439220522	0.05376282144890481	0.9981934408004447	0.500277932184547	0.022724287949754743
14	0.99854088137713	0.9983681889140577	0.011933922164693021	0.9999305169538633	0.500277932184547	0.0011226925517091137

Table C.15: LSTM 5-Fold Cross Validation Fold 5

epoch	acc	f1_score	loss	val_acc	val_f1_score	val_loss
0	0.76511166426375307	0.7054246919636232	0.4804014547419222	0.8668704836020011	0.4311501865190821	0.45728826068244405
1	0.8844690718963331	0.8696034708155431	0.2766824494656806	0.8419955530850473	0.346013189861981	0.3093970443058413
2	0.9088571974500165	0.8922793417234609	0.21505356937852385	0.7643135075041689	0.41024941931571346	0.7132558809555011
3	0.9563480345324741	0.9537908106638004	0.13490802633455362	0.997290161200667	0.5000977514849032	0.019331063177558765
4	0.9789643731869583	0.9774618058188996	0.07398786254155787	0.9986103390772652	0.4996073652864364	0.009079231832201296
5	0.9719293369695495	0.9699987488264762	0.0845567378576881	0.9921484157865481	0.4957283555699296	0.03733085890302125
6	0.9139467421702653	0.9037222100850199	0.21584309076078248	0.92898833268482491	0.4320053331980512	0.1625104482990253
7	0.9780611092775626	0.97722298388604026	0.07088468429322298	1.0	0.500277932184547	0.0005168934727891071
8	0.9763240632979555	0.9737505113068632	0.062400393260017915	0.8387298499166204	0.34169943084844023	0.5535133054902261
9	0.9888829057305147	0.98804825955570587	0.032787533267166546	0.9999305169538633	0.500277932184547	0.0008676509623935811
10	0.9985061404575379	0.9983090233830957	0.011459913337334877	1.0	0.500277932184547	8.973098257770759e-06

epoch	acc	f1'score	loss	val'acc	val'f1'score	val'loss
11	0.8703121471625354	0.8597507694397917	0.34133467095825665	0.8863952195664258	0.4055419420122504	0.2662167148875552
12	0.9735274192707881	0.9720675557984835	0.09322037019322307	1.0	0.500277932184547	0.00018618244611711996
13	0.9949973075787316	0.9943619618439604	0.029155275303406007	0.9998610339077265	0.5002426390783067	0.0010992790950630341
14	0.9964564262016016	0.9962320541622769	0.021537448640063475	1.0	0.500277932184547	2.199372740658756e-05

Appendix D

ANTENNA BEAM PATTERNS

Figure D.1: Time Domain Broadspec Antenna Azimuth Beam Pattern for (a) 3 GHz, (b) 4 GHz, (c) 5 GHz, and (d) 6 GHz. Image from [59].

Figure D.2: Time Domain BroadSpec Antenna Elevation Beam Pattern for (a) 3 GHz, (b) 4 GHz, (c) 5 GHz, and (d) 6 GHz. Image from [59].