

# AUS Repository

## Isolating Physical Replacement of Identical IoT Devices Using Machine and Deep Learning Approaches

Item Type	Thesis
Authors	Mohammad, Areej Osama
Download date	2026-05-19 10:28:29
Link to Item	<a href="http://hdl.handle.net/11073/21502">http://hdl.handle.net/11073/21502</a>

ISOLATING PHYSICAL REPLACEMENT OF IDENTICAL IOT DEVICES  
USING MACHINE AND DEEP LEARNING APPROACHES

by

Areej Osama Mohammad

A Thesis presented to the Faculty of the  
American University of Sharjah  
College of Engineering  
In Partial Fulfilment  
of the Requirements  
for the Degree of

Master of Science in  
Computer Engineering

Sharjah, United Arab Emirates

April 2021

## **Declaration of Authorship**

I declare that this thesis is my own work and, to the best of my knowledge and belief, it does not contain material published or written by a third party, except where permission has been obtained and/or appropriately cited through full and accurate referencing.

Signed: Areej Osama Mohammad

Date: 22 - April - 2021

The Author controls copyright for this report.

Material should not be reused without the consent of the author. Due acknowledgement should be made where appropriate.

© Year 2021

Areej Osama Mohammad

**ALL RIGHTS RESERVED**

## Approval Signatures

We, the undersigned, approve the Master's Thesis of Areej Osama Mohammad

Thesis Title: Isolating Physical Replacement of Identical IoT Devices Using Machine and Deep Learning Approaches

Date of Defense: 22 - April - 2021

<b>Name, Title and Affiliation</b>	<b>Signature</b>
Dr. Imran Zualkernan Professor, Department of Computer Science and Engineering Thesis Advisor	
Dr. Fadi Aloul Professor, Department of Computer Science and Engineering Thesis Co-Advisor	
Dr. Salam Dhou Assistant Professor, Department of Computer Science and Engineering Thesis Committee Member	
Dr. Usman Tariq Assistant Professor, Department of Electrical Engineering Thesis Committee Member	
Dr. Fadi Aloul Head Department of Computer Science and Engineering	
Dr. Lotfi Romdhane Associate Dean for Graduate Affairs and Research College of Engineering	
Dr. Sameer Al Asheh Interim Dean College of Engineering	
Dr. Mohamed El-Tarhuni Vice Provost for Graduate Studies Office of Graduate Studies	

## **Acknowledgement**

First and foremost, praises and thanks to God, the Almighty, for His graces and blessings, that helped me complete my thesis successfully.

I would like to express my deepest appreciation to my advisors, Dr. Imran Zualkeman and Dr. Fadi Aloul, who provided me with the knowledge, support, and encouragement throughout this research. I am deeply beholden for their immeasurable assistance, patience, worthy discussion, and suggestions, which without them, this work would have never been accomplished.

I would like to thank Dr. Salam Dhou and Dr. Usman Tariq for serving as examining committee and providing me with great guidance and significant input.

I would also like to thank the professors of the Computer Science and Engineering department who taught me the master level courses with expertise, abundant knowledge, and skills, that have been of great help to perform this thesis.

I would also like to thank the Graduate Program and the College of Engineering at the American University of Sharjah for offering me partial Graduate Teaching Assistantship which allowed me to continue my higher education in Computer Engineering.

To my mother, who always pushed me to continue doing my best, thank you for all your efforts and prayers during this thesis. I am extremely grateful to my family and friends for their wise counsel, support, and words of encouragement. To my fiancé, Islam Hourani, the most caring and supportive partner, your support, and motivation is something I will always be grateful for.

## **Dedication**

*This thesis is wholeheartedly dedicated to my beloved parents, who have been dedicating their life to raise us, who have been the source of moral, emotional, and financial support.*

## Abstract

Many Internet of Things (IoT) applications deploy identical end devices like sensor nodes or surveillance cameras in an organization. The purpose of this thesis was to determine if a malicious physical substitution of one end device by an identical compromised device could be recognized using deep learning or machine learning techniques. Conventional techniques to address the physical replacement of a node require one to implement specialized hardware like Physical Unclonable Functions (PUF) using expensive encryption techniques. For low-resource devices like an ESP32, the device does not come with an integrated PUF module, and a separate chip is required to execute the cryptographic operations. Moreover, recently deep learning techniques are shown to have compromised the PUF-based technique as well. This thesis explored whether machine and deep learning could be used to identify a single end device from a set of identical devices without requiring a PUF-like mechanism for authentication. Network data from 18 identical ESP32 devices was collected in a typical MQTT-based IoT network. In addition to exploring conventional machine learning methods including Random Forest, Bayesian, SVM, LightGBM, Gradient Boosting, and XGBoost, a tiny Convolutional Neural Network (CNN) was designed and optimized using the Hyperband algorithm. The CNN was small with 85,058 trainable parameters and only used the packet Inter-Arrival Time (IAT) as input. The results are that the CNN outperformed all other models, with a micro-F1-Score of 0.999 (0.0012). The Random Forest model was the best traditional machine learning models with a micro-F1-Score of 0.9501 (0.0036). The worst was Bayesian with a micro-F1-Score of 0.222 (0.0016). There was a statistically significant difference in the F1-Score (Kruskal-Wallis; chi-square=84.192,  $p < 0.05$ ) between the various trained model using 10-fold validation.

**Keywords: IoT Edge Devices; Inter-Arrival Time, Machine Learning, Convolutional Neural Networks.**

## Table of Contents

Abstract .....	6
List of Figures .....	10
List of Tables .....	13
List of Abbreviations.....	17
Chapter 1. Introduction.....	18
1.1. Overview .....	18
1.2. Thesis Definition.....	19
1.3. Research Contribution .....	19
1.4. Thesis Organization.....	19
Chapter 2. Background.....	20
2.1. Internet of Things.....	20
2.2. ESP Espressif Edge Device.....	21
2.3. Message Queuing Telemetry Transport Protocol.....	23
2.4. Device Fingerprinting.....	24
2.5. Deep Learning.....	25
Chapter 3. Related Work.....	27
3.1. IoT Device Traditional Authentication Methods .....	27
3.1.1. Encryption-based authentication techniques.....	27
3.1.2. Password-based authentication techniques.....	28
3.1.3. Challenge-Response authentication technique.....	28
3.2. Fingerprinting Authentication Techniques .....	28
3.2.1. Fingerprinting using classical methods. ....	29
3.2.2. Fingerprinting using machine learning. ....	31
Chapter 4. Methodology.....	46
4.1. Use Case Scenario .....	46
4.2. Data Description.....	47

4.3.	Data Preprocessing.....	49
4.4.	Model Building.....	52
4.5.	Models Evaluation Criteria .....	53
4.5.1.	Confusion matrix.....	53
4.5.2.	Evaluation metrics.....	54
Chapter 5.	Experimental Design: Setup and Data Gathering.....	57
5.1.	Experimental Setup .....	57
Chapter 6.	Traditional Machine Learning Models: Experiments and Results ...	60
6.1.	Data Processing.....	60
6.2.	Machine Learning Models .....	61
6.2.1.	Random forest model.....	62
6.2.2.	Bayesian model.....	70
6.2.3.	Support vector machine model.....	76
6.2.4.	LightGBM model.....	92
6.2.5.	Gradient boost model.....	100
6.2.6.	XG-Boost model.....	109
Chapter 7.	Convolutional Neural Networks: Experiment 1.....	118
7.1.	Convolutional Neural Networks.....	118
7.2.	Input to the Convolutional Neural Network.....	120
7.3.	CNN Architecture and Results.....	123
7.4.	Hypertuning.....	129
7.5.	Discussion.....	135
7.5.1.	Convolutional layer 1.....	136
7.5.2.	Convolutional layer 2.....	137
7.5.3.	Convolutional layer 3.....	142
Chapter 8.	Convolutional Neural Network – Experiment 2 .....	148
8.1.	Input, Model, and Results of CNN from Literature .....	148

8.2	Input to the Model and Model Fitting .....	149
Chapter 9.	Discussion of Model Results.....	153
9.1.	Summary of Model Results.....	153
Chapter 10.	Conclusion and Future Work .....	155
References	.....	156
Appendix A	.....	165
Appendix B	.....	168
Appendix C	.....	177
Appendix D	.....	182
Vita	.....	185

## List of Figures

Figure 2.1: Internet of Things Layers [2].....	21
Figure 2.2: ESP8266 Device [7].....	22
Figure 2.3: ESP32 SparkFun Device [9].....	23
Figure 2.4: Publish Subscribe Model [12].....	24
Figure 2.5: Neural Networks Feature Extraction Approaches.....	25
Figure 2.6: Deep Learning Illustration [14].....	25
Figure 3.1: Autoencoders [40].....	32
Figure 4.1: Use Case Scenario for a Surveillance Camera in a Smart Home.....	47
Figure 4.2: Statistical Features of Data.....	50
Figure 4.3: Mel Values of Data.....	51
Figure 4.4: Confusion Matrix.....	53
Figure 4.5 ROC Curve.....	55
Figure 4.6: K-Fold with 5 Folds Example.....	56
Figure 5.1: Lab Setup.....	58
Figure 5.2: Experimental Setup.....	59
Figure 6.1: Sample Statistical Data for Traditional Machine Learning Models.....	61
Figure 6.2 Random Forest Classifier [93].....	62
Figure 6.3: Confusion Matrix for Random Forest (Statistical Input). ....	64
Figure 6.4: ROC Curve for Random Forest Classifier (Statistical Input). ....	64
Figure 6.5: Confusion Matrix (Mel Input) for Random Forest. ....	68
Figure 6.6: ROC Curve for Random Forest (Mel Input) Classifier.....	68
Figure 6.7: Confusion Matrix for Bayesian (Statistical Input).....	72
Figure 6.8: ROC Curve for Bayesian Classifier (Statistical Input). ....	72
Figure 6.9: Confusion Matrix (Mel Input) for Bayesian Classifier. ....	75
Figure 6.10: ROC Curve for Bayesian Classifier (Mel Input). ....	76
Figure 6.11: SVM Example [95].....	77
Figure 6.12: Confusion Matrix for SVM – Linear (Statistical Input).....	79
Figure 6.13: ROC Curve for SVM Linear Classifier (Statistical Input). ....	79
Figure 6.14: Confusion Matrix (Mel Input) for SVM Linear. ....	83
Figure 6.15: ROC Curve for SVM Linear Classifier (Mel Input). ....	83
Figure 6.16: Confusion Matrix for SVM - Non-Linear (Statistical Input).....	86

Figure 6.17: ROC Curve for SVM Non-Linear Classifier (Statistical Input).....	87
Figure 6.18: Confusion Matrix for SVM Non-Linear (Mel Input).....	90
Figure 6.19: ROC Curve for SVM Non-Linear Classifier (Mel Input). ....	90
Figure 6.20: Confusion Matrix for LightGBM (Statistical Input).....	94
Figure 6.21: ROC Curve for LightGBM (Statistical Input).....	94
Figure 6.22: Confusion Matrix for LightGBM (Mel Input).....	98
Figure 6.23: ROC Curve for LightGBM (Mel Input).....	98
Figure 6.24: Gradient Boost Algorithm Example [100].....	100
Figure 6.25: Confusion Matrix for Gradient Boost (Statistical Input).....	102
Figure 6.26: ROC Curve for Gradient Boost Classifier (Statistical Input). ....	103
Figure 6.27: Confusion Matrix for Gradient Boost (Mel Input).....	107
Figure 6.28: ROC Curve for Gradient Boost Classifier (Mel Input).....	107
Figure 6.29: Confusion Matrix for XG-Boost (Statistical Input).....	111
Figure 6.30: ROC Curve for XG-Boost Classifier (Statistical Input).....	111
Figure 6.31: Confusion Matrix for XG-Boost (Mel Input).....	115
Figure 6.32: ROC Curve for XG-Boost Classifier (Mel Input).....	115
Figure 7.1 Convolutional Layer [102].....	119
Figure 7.2: Max and Average Pooling .....	119
Figure 7.3: Fully-Connected Layer.....	120
Figure 7.4: CNN Architecture.....	120
Figure 7.5: Mel Banks Plot.....	122
Figure 7.6: Device 1 Time and Mel Plots. ....	123
Figure 7.7: CNN Model Before Hypertuning.....	124
Figure 7.8: CNN Architecture Before Hypertuning.....	124
Figure 7.9: Training vs Validation Loss Before Hypertuning.....	126
Figure 7.10: Confusion Matrix for Model Before Hypertuning.....	127
Figure 7.11: ROC Curve Before Hypertuning.....	128
Figure 7.12: CNN Model After Hypertuning. ....	130
Figure 7.13: CNN Model Architecture After Hypertuning.....	130
Figure 7.14: Training versus Validation Loss After Hypertuning.....	132
Figure 7.15: CNN Model ROC Curve. ....	133
Figure 7.16: Fold 3 Confusion Matrix. ....	135
Figure 7.17: Convolutional Layer 1.....	136

Figure 7.18: Filters Distribution for Convolutional Layer 1.....	137
Figure 7.19: Convolutional Layer 2.....	137
Figure 7.20: Layer 2 Filters 1 – 30. ....	138
Figure 7.21: Layer 2 Filters 31 – 63. ....	139
Figure 7.22: Optimum Number of K-Means Clusters.....	140
Figure 7.23: K-Means Cluster Means Plot.....	141
Figure 7.24: Feature Map of Layer 2 Filter on Device 0.....	142
Figure 7.25: Feature Map of Layer 2 Filter on Device 11. ....	142
Figure 7.26: Convolutional Layer 3.....	143
Figure 7.27: Layer 3 Filters 1 - 24 Histogram.....	143
Figure 7.28: Layer 3 Filters 25 - 48 Histogram. ....	144
Figure 7.29: Layer 3 Filters 49 - 64 Histogram. ....	145
Figure 7.30: Entropy Distribution for Filters in Third Layer.....	146
Figure 7.31: Filter 8 Distribution.....	146
Figure 8.1: Dense Model Architecture [71].....	148
Figure 8.2: Confusion Matrix [71].....	149
Figure 8.3: Model Architecture [74].....	150
Figure 8.4: Training Loss vs Validation Loss Curve (Binary Input).....	150
Figure 8.5: Confusion Matrix for CNN (Binary Input). ....	152

## List of Tables

Table 2.1: ESP8266 Specifications.....	21
Table 2.2: ESP32 Specifications. ....	22
Table 3.1: Models Result in [53].....	34
Table 3.2: Accuracy of Model per Service in [54].....	34
Table 3.3: Models Result in [55].....	35
Table 3.4: Precision Results for the Models in [57].....	36
Table 3.5: Accuracy and Precision Results for the Models in [58]. ....	36
Table 3.6: Accuracy of Bonsai Model in [59].....	37
Table 3.7: Accuracy of Models in [60].....	37
Table 3.8: Model Results in [62].....	38
Table 3.9: CNN and other Model Results in [64].....	39
Table 3.10: List of Fingerprinting Features in [69].....	40
Table 3.11: Results of DBSCAN Model in [71].....	41
Table 3.12: Summary of Authentication using Machine Learning – Part 1.....	42
Table 3.13: Summary of Authentication using Machine Learning – Part 2.....	43
Table 3.14: Summary of Authentication using Machine Learning – Part 3.....	44
Table 3.15: Summary of Authentication using Machine Learning - Cont'd. ....	45
Table 4.1: Network Layer Features. ....	47
Table 4.2: Transmission Layer Features. ....	48
Table 4.3: MQTT Features.....	48
Table 4.4: Data Summary.....	49
Table 4.5: Experiment Types.....	52
Table 5.1: MQTT Commands.....	58
Table 5.2: MQTT Topics.....	58
Table 6.1: Random Forest Experiment Parameters.....	62
Table 6.2: Random Forest Classification Results (Statistical Input).....	63
Table 6.3: Random Forest Classifier (Statistical Input) AUC Values.....	65
Table 6.4: K-Fold for Random Forest (Statistical Input) - Mean (Standard Dev.).....	66
Table 6.5: Random Forest (Statistical Input) Hypertuning.....	66
Table 6.6: K-Fold for (Statistical Input) Random Forest - Mean (Standard Dev.) After Hypertuning. ....	66

Table 6.7: Random Forest (Mel Input) Classification Results.....	67
Table 6.8: K-Fold for Random Forest (Mel Input) - Mean (Standard Dev.).....	69
Table 6.9: Random Forest (Mel Input) Hypertuning.....	69
Table 6.10: K-Fold for Random Forest (Mel Input) - Mean (Standard Dev.) After Hypertuning.....	69
Table 6.11: Bayesian Classifier (Statistical Input) Experiment Parameters.....	70
Table 6.12: Bayesian Classifier (Statistical Input) Classification Results.....	70
Table 6.13: Bayesian Classifier (Statistical Input) AUC Values.....	73
Table 6.14: K-Fold for Bayesian (Statistical Input) - Mean (Standard Deviation).....	74
Table 6.15: Bayesian Classifier (Mel Input) Classification Results.....	74
Table 6.16 K-Fold for Bayesian (Mel Input) - Mean (Standard Deviation).....	76
Table 6.17: SVM Linear Classifier Experiment Parameters (Statistical Input).....	77
Table 6.18: SVM Linear Classifier Classification Results (Statistical Input). ....	78
Table 6.19: SVM Linear Classifier AUC Values (Statistical Input). ....	80
Table 6.20: K-Fold for SVM Linear Classifier (Statistical Input) - Mean (Standard Deviation).....	81
Table 6.21: SVM Linear Classifier (Statistical Input) Hypertuning.....	81
Table 6.22: K-Fold for SVM Linear (Statistical Input) - Mean (Standard Deviation) After Hypertuning. ....	81
Table 6.23: SVM Linear Classifier (Mel Input) Classification Results.....	82
Table 6.24: K-Fold SVM Linear Classifier (Mel Input)- Mean (Standard Deviation)84	
Table 6.25: SVM Linear Classifier (Mel Input) Hypertuning.....	84
Table 6.26 K-Fold for SVM Linear (Mel Input) - Mean (Standard Deviation) After Hypertuning. ....	84
Table 6.27: SVM Non-Linear Classifier Experiment Parameters (Statistical Input)..	85
Table 6.28: SVM Non-Linear Classifier Classification Results (Statistical Input). ...	85
Table 6.29: SVM Non-Linear Classifier AUC Values (Statistical Input).....	87
Table 6.30: K-Fold for SVM Non-Linear Classifier (Statistical Input) - Mean (Standard Deviation).....	88
Table 6.31: SVM Non-Linear Classifier (Statistical Input) Hypertuning.....	88
Table 6.32: K-Fold for SVM Non-Linear (Statistical Input) - Mean (Standard Deviation) After Hypertuning. ....	88
Table 6.33: SVM Non-Linear Classifier Experiment Parameters (with Mel Input)..	89

Table 6.34: SVM Non-Linear Classifier Classification Results (Mel Input). .....	89
Table 6.35: K-Fold for SVM Non-Linear Classifier (Mel Input) - Mean (Standard Deviation).....	91
Table 6.36: SVM Non-Linear Classifier (Mel Input) Hypertuning.....	91
Table 6.37: K-Fold for SVM Non-Linear (Mel Input) - Mean (Standard Deviation) After Hypertuning. ....	91
Table 6.38: LightGBM Classifier (Statistical Input) Classification Results. ....	93
Table 6.39: LightGBM Classifier (Statistical Input) AUC Values.....	95
Table 6.40: K-Fold for LightGBM Classifier (Statistical Input) - Mean (Standard Deviation).....	96
Table 6.41: LightGBM Classifier (Statistical Input) Hypertuning.....	96
Table 6.42: K-Fold for LightGBM (Statistical Input) - Mean (Standard Deviation) After Hypertuning. ....	96
Table 6.43: LightGBM Classifier (Mel Input) Classification Results. ....	97
Table 6.44: K-Fold LightGBM Classifier (Mel Input) -Mean (Standard Deviation)..	99
Table 6.45: LightGBM Classifier (Mel Input) Hypertuning.....	99
Table 6.46: K-Fold for LightGBM (Mel Input) - Mean (Standard Deviation) After Hypertuning. ....	99
Table 6.47: Gradient Boost (Statistical Input) Classifier Experiment Parameters....	101
Table 6.48: Gradient Boost Classifier (Statistical Input) Classification Results.....	101
Table 6.49: Gradient Boost Classifier AUC Values (Statistical Input).....	103
Table 6.50: K-Fold Gradient Boost (Statistical Input) - Mean (Standard Deviation)104	
Table 6.51: Gradient Boost Classifier (Statistical Input) Hypertuning.....	104
Table 6.52: K-Fold for Gradient Boost (Statistical Input) - Mean (Standard Deviation) After Hypertuning. ....	105
Table 6.53: Gradient Boost (Mel Input) Classifier Experiment Parameters. ....	105
Table 6.54: Gradient Boost Classifier (Mel Input) Classification Results.....	106
Table 6.55: K-Fold for Gradient Boost (Mel Input) - Mean (Standard Deviation)..	108
Table 6.56: Gradient Boost Classifier (Mel Input) Hypertuning.....	108
Table 6.57: K-Fold for Gradient Boost (Mel Input) - Mean (Standard Deviation) After Hypertuning. ....	108
Table 6.58: XG-Boost Classifier Experiment Parameters. ....	109
Table 6.59: XG-Boost Classifier Classification Results (Statistical Input).....	110

Table 6.60: XG-Boost Classifier AUC Values (Statistical Input).....	112
Table 6.61: K-Fold (Statistical Input) for XG-Boost Classifier - Mean (Standard Deviation).....	113
Table 6.62: XG-Boost Classifier (Statistical Input) Hypertuning .....	113
Table 6.63: K-Fold for XG-Boost (Statistical Input) - Mean (Standard Deviation) After Hypertuning. ....	113
Table 6.64: XG-Boost Classifier Experiment Parameters.....	114
Table 6.65: XG-Boost Classifier (Mel Input) Classification Results.....	114
Table 6.66: K-Fold (Mel Input) XG-Boost Classifier - Mean (Standard Deviation).116	
Table 6.67: XG-Boost Classifier (Mel Input) Hypertuning.....	116
Table 6.68: K-Fold for Gradient Boost (Mel Input) - Mean (Standard Deviation) After Hypertuning. ....	116
Table 6.69: F1-Score Results for 10-Folds Runs for all Models – (Mean and Standard Deviation).....	117
Table 7.1: CNN Before Hypertuning Classification Report.....	127
Table 7.2: CNN Model Before Hypertuning AUC Values.....	128
Table 7.3: Hypertuning Results.....	129
Table 7.4: CNN After Hypertuning Classification Report. ....	133
Table 7.5: CNN Model AUC Values.....	134
Table 7.6: K-Fold Results for Model After Hypertuning. ....	135
Table 7.7: K-Mean Clustering Result.....	140
Table 7.8: Entropy Values for Filter in Third Layer. ....	147
Table 8.1: Classification Report for CNN (Binary Input). ....	151
Table 9.1: Results for all Models on K-Fold – Mean (Standard Deviation).....	153
Table 9.2: Pairwise Wilcoxon Test Results (p-values).....	154

## List of Abbreviations

ANN	Artificial Neural Network
AUC	Area Under Curve
CNN	Convolutional Neural Network
EFB	Exclusive Feature Bundling
IAT	Inter-Arrival Time
IoT	Internet of Things
K-NN	K-Nearest Neighbor
LightGBM	Light Gradient Boosting Machine
LSTM	Long-Short Term Memory
MFCC	Mel Frequency Cepstral Coefficients
MQTT	Message Queue Telemetry Transport
OTP	One Time Passwords
PUF	Physical Unclonable Functions
RBF	Radial Basis Function
RNN	Recurrent Neural Networks
ROC	Receiver Operating Characteristic
RTOS	Real Time Operating System
SVM	Support Vector Machine
XG-Boost	Extreme Gradient Boosting

## Chapter 1. Introduction

In this chapter, we provide a short introduction about the IoT physical devices security challenges. Then, we present the problem investigated in this study as well as the thesis contribution. Finally, general organization of the thesis is presented.

### 1.1. Overview

The huge rise in the number of connected IoT devices has led to several security risks in achieving the authenticity of the physical devices. One of the most challenging risks is ensuring that a device has not been physically replaced by another identical device. By identical, this refers to replacing a device with another one device that comes from the same vendor, with the same type, running the same OS and application, and containing the same network information such as IP and MAC address. In other words, the software stack is of the original and the malicious one is identical, making the illegitimate device identification less obvious. If one of the devices were replaced by an identical device, hackers can execute malicious code, redirect traffic, and spam the entire network and the behaviour of the environment. As far as the literature work is concerned, the work looked at was mostly on how devices can be protected from remote attacks that affect its *behaviour*, overlooking the physical device replacement problem.

To differentiate between the two IoT devices, certain features from the devices must be leveraged to perform the classification. Many studies have used features by capturing information at the application level, network level, or the sensing level. Then, machine learning techniques such as Deep Neural Networks, Naïve Bayes, Random forest, can be used to classify and identify between the devices. In this way, the features are used to uniquely identify the device, and machine learning models are used to ensure that using these features, one can detect if a device has been physically replaced.

One approach to such a problem is by creating fingerprints for every device that uniquely identify it. The features selected from the devices will be used to generate such fingerprints. These fingerprints will be generated using features from the application, network or sensing layers or a combination of them. This fingerprint will be leveraged by the identification method to uniquely identify the device. The identification methods in the literature vary and are discussed in Chapter 3. In this thesis, a single feature; Interarrival time will be considered to identify between the

devices, with multiple approaches. A CNN will be built and trained using this feature to be able to uniquely identify between the edge devices. This work will be compared with the traditional machine learning models.

### **1.2. Thesis Definition**

Driven by the extreme adoption of the IoT technology and their devices in different applications, we will focus on the problem of accurately detecting if a physical IoT device has been illegitimately replaced by another device. That is; *given a smart IoT environment, a feature from the IoT devices is leveraged to train a deep learning model, to ensure device replacement by an identical one is accurately detected. The feature selected in this thesis is the **Inter-Arrival Time** between the network packets.* The objective of this work is to accurately identify between the identical IoT devices in a critical smart environment such as a hospitals, financial entities, and others by leveraging deep neural networks. For example, in an environment that contains identical surveillance cameras like in a hospital, or environments that contain identical sensors for fire detection, the work should be able to identify if any of these devices has been illegitimately replaced by a faulty one.

### **1.3. Research Contribution**

The thesis work contribution can be summarized as follows. Leveraging Artificial Neural Networks to uniquely identify if an IoT edge device has being physically replaced by another identical one, running the same OS type, application, containing the same network identification information such IP, MAC addresses.

### **1.4. Thesis Organization**

The rest of the thesis is organized as follows. Chapter 2 provides background about IoT and device fingerprinting for authentication. Chapter 3 discusses the related works to this research. Chapter 4 describes the address the problem and explains the data pre-processing phase. Chapter 5 details the experimental set up done in this research work. In the following three Chapters, the experiments performed. This is presented in Chapter 6 that contains the traditional machine learning models, Chapter 7 that contains the CNN model, and Chapter 8 contains the CNN comparison with literature. Chapter 9 provides a summary for all models. Finally, Chapter 10 concludes the thesis and outlines the future work.

## Chapter 2. Background

This chapter briefly introduces the concept of Internet of Things (IoT). Next, this chapter describes the edge devices (ESP), and the commonly run protocol Message Queue Telemetry Transport (MQTT) used in this thesis. Then, it introduces the concept of device fingerprinting. Finally, it examines current work and research done edge device fingerprinting.

### 2.1. Internet of Things

Internet of Things (IoT) refers to a system of objects connected to each other and communicate via the Internet. These objects can be devices such as sensors, smartphones, cameras, or any wearable [1]. It also includes any “thing” that allows it to be identified in the Internet world and has the ability to communicate with other devices. These things communicate with each other to monitor, control, operate smart environments in different use cases. This communication produces huge amounts of data per time transmitted between the devices and the servers. IoT systems have been deeply incorporated in multiple environments such as is in smart homes, smart cities, smart factories, smart workplaces, smart retail, etc.

There are different paradigms IoT can be looked at. In terms of the architecture, IoT architecture can be summarized into four layers: *sensing (or perception) layer*, *networking layer*, *middleware layer*, and *the application layer* [2]. The four layer architecture is presented in Figure 2.1. The sensing layer contains the underlying physical devices in the network that actually run the application logic. These include sensors, embedded devices, microcontrollers, or actuators. The networking layer will run all the network related functions from routing data back and forth. In IoT, this contains the routers, gateways, running communication protocols such as Wi-Fi, 3G, 4G [3]. The middleware layer provides the services that IoT devices will consume to perform functionalities. Examples of these services include storage, communication, location, data collection, authentication, authorization, event processing, device management services, etc. Finally, the application layer determines a set of protocols for message passing at the application level. Most familiar application protocols for IoT devices are: ZigBee, XMPP, Message Queuing Telemetry Transport (MQTT), Constrained Application Protocol (CoAP), 6LowPAN, Lora WAN [4] etc.

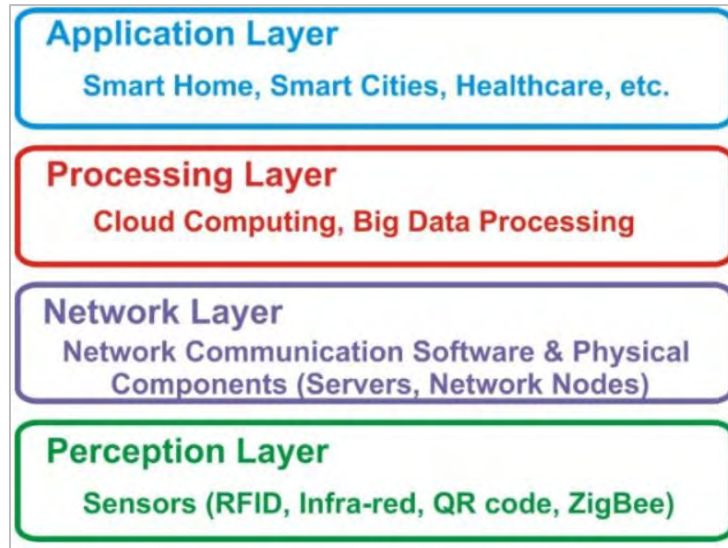


Figure 2.1: Internet of Things Layers [2].

## 2.2. ESP Espressif Edge Device

ESP edge devices are a family of devices developed by Espressif [5]. Espressif community develops Wi-Fi and Bluetooth, low-power IoT solutions and devices. The most popular of their devices are ESP8266 and ESP32 series of chips, modules and development boards. Table 1 summarizes their main features [6], and Figure 2.2 presents the ESP8266 SparksFun board.

Table 2.1: ESP8266 Specifications.

Specification	ESP8266
MCU	Xtensa Single Core 32-bit
802.11 b/g/n Wi-Fi	Yes
Bluetooth	No
Frequency	80 MHz
SRAM	160 KBytes
Flash	SPI Flash, up to 16 MBytes
GPIO	17
ADC	10 Bits
Ethernet MAC Interface	No
Touch Sensor	No
Temperature Sensor	No
Working Temperature	- 40 degrees to 125 degrees
Cost	\$ 16.95

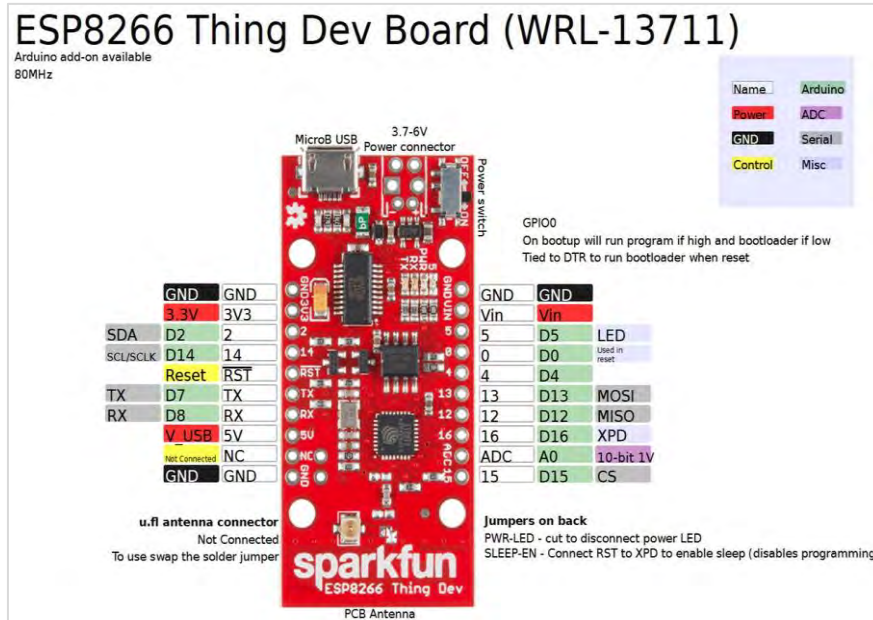


Figure 2.2: ESP8266 Device [7].

Table 2.2 summarizes the main features of ESP32 [8], and Figure 2.3 presents the ESP8266 SparksFun board.

Table 2.2: ESP32 Specifications.

Specification	ESP32
MCU	Xtensa Dual Core 32-bit
802.11 b/g/n Wi-Fi	Yes
Bluetooth	Yes
Frequency	160 MHz
SRAM	512 KBytes
Flash	SPI Flash, up to 16 Mbytes
External Memory	Up to 1 GB of external flash support
Power	5 uA current, the chip automatically powers on/off RF transceiver when needed
GPIO	36
ADC	12 Bits
Ethernet MAC Interface	Yes
Touch Sensor	Yes
Temperature Sensor	Yes
Working Temperature	- 40 degrees to 125 degrees
Cost	\$ 20.95

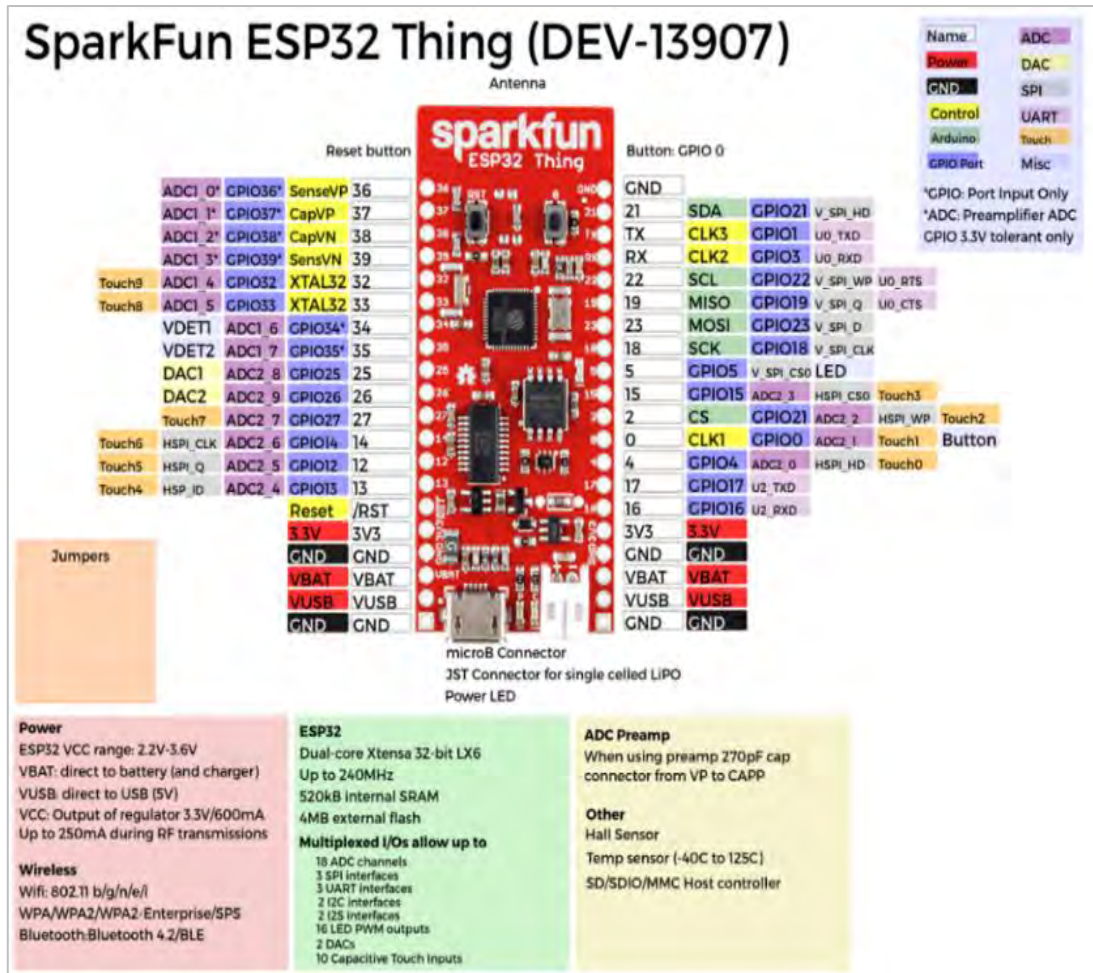


Figure 2.3: ESP32 SparkFun Device [9].

ESP8266 and ESP32 can run variety of Real-Time Operating Systems (RTOS), such as Mongoose OS, FreeRTOS, Arduino Core, NodeMCU, Lua, etc [10]. Due to its recent adoption in IoT projects and research, this board will be used to build and simulate the smart IoT environment we will be looking at. The selected chip is ESP32 and will be the IoT edge devices.

### 2.3. Message Queuing Telemetry Transport Protocol

MQTT is an application layer messaging protocol, designed for low power and low memory footprint communication. MQTT leverages the publish-subscribe messaging protocol to communicate messages between the connected parties. An illustration of the publish-subscribe protocol is presented in Figure 2.4. Every device will publish and subscribe to the desired “topic”. A broker receives all messages, filters them, determines who is subscribed to each message, and sends the message to the subscribed clients. The MQTT protocol is based on TCP/IP [11].

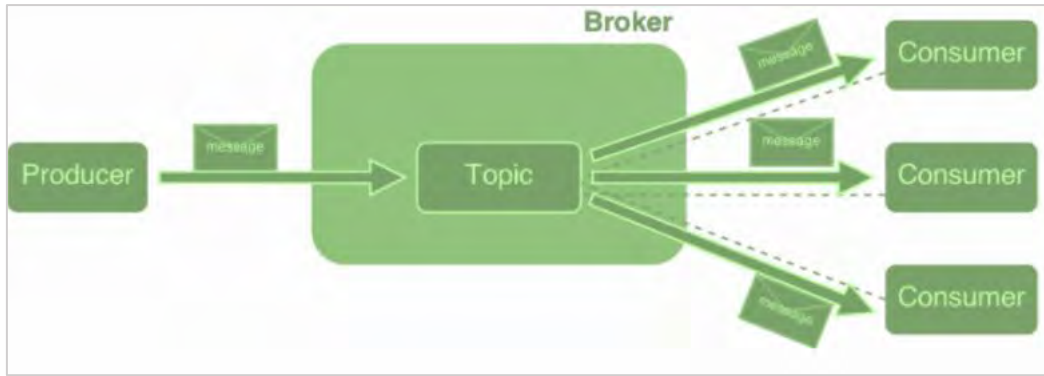


Figure 2.4: Publish Subscribe Model[12].

## 2.4. Device Fingerprinting

Device fingerprinting is a technique used to identify a device connected to the Internet. As human beings are authenticated by their unique biometric fingerprints, devices also contain unique information that uniquely identify them. Whilst this information can be very challenging to extract, especially with the large number of connected devices, many different device information has been leveraged to create the device fingerprints. As for IoT devices, device fingerprinting is important to ensure that a device has been illegitimately replaced by another one. Studies have proposed unique device identification by generating fingerprints from features at the application level, network level, or the sensing level. For example, some of these features used to generate the fingerprints include features specific to application layer type, packet counts, size, interarrival time between packets, the entropy of payloads, count of packets, physically unclonable functions from device, delay between packets and others.

In an IoT environment, these features are produced from the data generated from the devices. There are two parts in this problem: *first, the feature extraction, and second, generating the device fingerprinting from these features*. As for the feature extraction, there are two approaches to perform this. First, the features that will uniquely identify the device is already decided and known, and fingerprints will be generated from them. Second, the other approach would be involving leverages advanced machine learning techniques such as neural networks to extract these features and generate the unique fingerprints. A summary for the approach is shown in Figure 2.5.

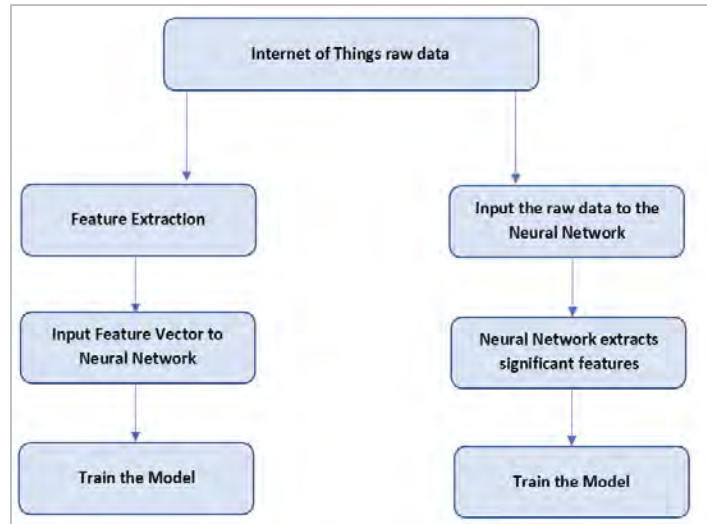


Figure 2.5: Neural Networks Feature Extraction Approaches.

## 2.5. Deep Learning

Deep learning is part of a broader family of machine learning methods based on Artificial Neural Networks. Unlike classical machine learning models, deep learning does not require features to be extracted beforehand, instead it will extract the most significant features by itself. Deep learning uses multiple neural network layers to progressively extract higher level features from the raw input [13]. An illustration of a deep learning neural network is presented in Figure 2.6.

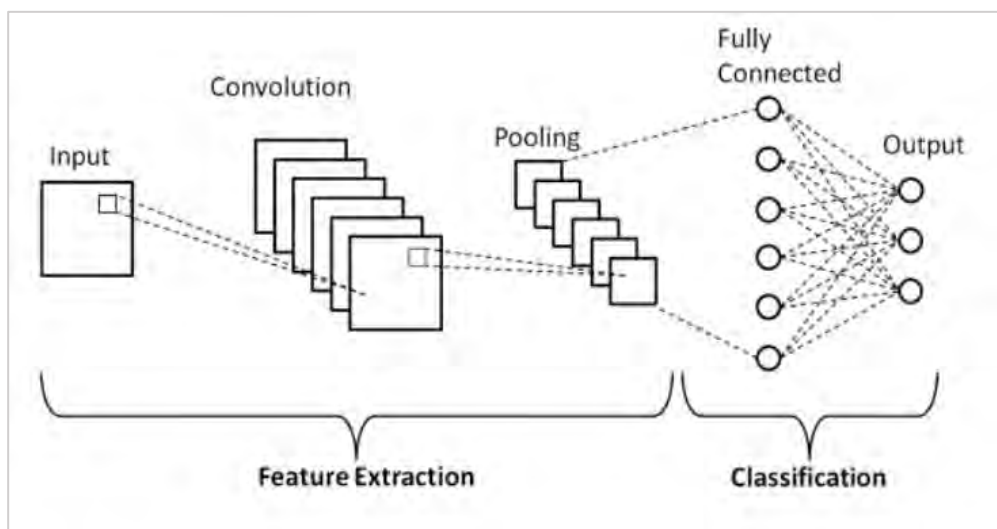


Figure 2.6: Deep Learning Illustration [14].

It does not contain limit to what it can learn, the more data it is fed, the more it can learn and the more accurate it will identify. However, deep learning contains some constraints:

- They typically require large amounts of data.

- Deep learning requires substantial amount computing power. Typically, deep learning models require high-performance GPUs that have a parallel architecture to run.

The learning (feature extraction) in deep learning can be supervised, unsupervised, or semi-supervised. The difference between the three approaches is summarized [15]:

- **Supervised:** all the data that is inserted to the model must be labelled. That is, we know beforehand that for every given data point in the dataset, it belongs to which class.
- **Unsupervised:** all the data that is inserted to the model is not labelled. That is, we do not know beforehand that data points in the dataset belongs to which class.
- **Semi-supervised:** the data that is inserted to the model can be labelled and not labelled.

## Chapter 3. Related Work

In this section, the different main techniques undertaken in the literature to tackle the problem of IoT physical device identification are discussed. This is organized in this section as follows. The first approach describes the traditional authentication techniques which are: encryption-based techniques, followed by the password-based ones, and challenge-response methods. Next approach presents the fingerprinting methods. Under the fingerprinting methods, we further classify into methods under; classical fingerprinting methods, fingerprinting using and autoencoders, fingerprinting using traditional machine learning models, and fingerprinting using Artificial Neural Networks. The work under machine learning methods is classified into supervised and unsupervised learning.

### 3.1. IoT Device Traditional Authentication Methods

In this section, we describe the main idea of each of the authentication techniques. In [16], a detailed survey of the literature work on these techniques is presented.

**3.1.1. Encryption-based authentication techniques.** In this method, symmetric and asymmetric cryptographic techniques are leveraged to authenticate a device. A unique encryption/decryption key is generated for every device. In methods that use a single key (symmetric encryption), the IoT device encrypts the authentication message using this key and the server decrypts the message using the same key. Upon the successful decryption, the server can authenticate the device [17]. In methods that leverage two keys; public and private keys (asymmetric encryption), digital signature technique is used to authenticate the device. In this technique, a digital signature consists of an encrypted hash of the original message. The IoT device hashes the authentication message and then encrypts it using its own private key and sends the result along with the original message. The server authenticates the device by performing the reverse process using the device public key. Upon successful signature verification, the server authenticates the device [18]. Despite these techniques are widely and successful used in authenticate schemes, their processes consume many resources which can be sometimes limited to resource constraint IoT devices.

**3.1.2. Password-based authentication techniques.** This method applies to the case when a human user wants to access an IoT device. The privileged user must configure the device with a certain password for it to use later, and manually enter it when needed to be accessed. These methods usually maintain a certain lifetime of a password before it is renewed [19]. To increase the privacy of the credential, some methods involve the user encrypting the credential before it is sent and decrypted at the server side [20]. Other advanced techniques [21] incorporate One-Time Passwords (OTP) that involves the user's trusted and own device to permit the access. Other methods leverage biometric information of the user to permit access [22]. Some other methods also encrypt and hashes the biometric information before it is transmitted for authentication [23]. While this authentication methods occur across the cloud, tampering and forging the passwords can easily happen [24].

**3.1.3. Challenge-Response authentication technique.** In this technique, the methods use a challenge-response architecture to authenticate a device. In this technique, the IoT device and the server both share a password. An IoT device can authenticate itself to the server by proving that it knows the password without revealing it [25]. The server that is willing to authenticate an IoT device generates a random challenge. The challenge is a nonce generated from a set seed. The IoT device will generate a response by applying a cryptographic function on the challenge received from the server and the stored password at the IoT device side. The IoT device will transmit the resultant response and along with it the challenge first received from the server. The server will perform the reverse operation to its initial received challenge. Upon successful match with the response received from the IoT device, the server can successfully authenticate the device [26]. Other authentication techniques [27] combine the challenge-response with Physically Unclonable Functions (PUF) methods. In these techniques, the challenge is a value generated from a PUF in the device hardware.

## **3.2. Fingerprinting Authentication Techniques**

Another approach to the authentication problem is by extracting features from IoT devices that uniquely identifies it. These are features can be at the application level, network level, sensing layer, or a combination. In this section we classify the approaches taken in the literature to perform fingerprinting.

**3.2.1. Fingerprinting using classical methods.** In this approach, techniques rely on generating fingerprints from devices using pre-determined features are used. For each device, a database of the fingerprints is stored. Whenever a device is to be authenticated, the fingerprint is compared by matching the generated fingerprint with the one stored. Typically, in such methods, a Hamming distance between the device and its stored signature is used for comparison to allow some room for errors.

Some methods involved uniquely identify embedded devices that contain SRAM using Physically Unclonable Functions, PUF. In [28], every device that contains SRAM has a unique ID generated from the on-board SRAM memory. The bits are generated randomly because of the inherently uncontrollable and unpredictable variations during the manufacturing process. To address the unpredictable aging and temperature variation that can alter the PUF, Hamming distance was used to authenticate the ID. The Probabilities of matching ID's is the typical measure taken to authenticate a device fingerprint.

In [29], Wallrabenstein proposed an algorithm that secures IoT devices by combining cryptography and PUF. The device security measures included device enrolment, authentication, producing the digital signatures, encryption and decryption were performed using a PUF based on an elliptic curve algorithm. The algorithm was tested on a Xilinx Artix 7 FPGA device, and the performance measure was time to complete the execution of the algorithm versus correct device authentication. The results complete each cryptographic operation in under 300 milliseconds for  $\approx 192$  symmetric key bits.

In [30], M. Barbareschi et al. proposed a scheme that eliminates the need to store the challenge and response pairs associated with traditional PUF authentication methods. In the traditional authentication methods that leverages PUF, a pair of challenge and response is stored and exchanged during the authentication or verification process, which introduces more storage and overhead to the device. In the proposed model, the challenge and response pair for the devices are stored in a set, off the device, and made available in an encrypted manner. The scheme was tested on Xilinx Zynq-7000 Field Programmable Gate Array. The performance measured considered were the overhead versus correct authentication. The results show that overhead was 1.44%.

In [31], W. Liang et al. proposed a bi-directional PUF-based RFID protocol to authenticate devices. The protocol allows the RFID tag and a server to both authenticate each other. The method uses a certain character padding operation along with a logical exclusive-OR (XOR) operation to generate the fingerprint from the PUF, i.e., the PUF response. In order to authenticate the device, the strings generated from the RFID tags are compared. The conclusions made that the proposed design contained reduced area, higher randomness, and higher stability compared to the conventional RFID PUF-based authentication methods.

B. Srinivasu et al. in [32] proposed an IoT authentication scheme using PUF based on Linear Feedback Shift Register (LFSR). The scheme leveraged two PUFs. The response from the first challenge associated with the first PUF is fed into the next PUF as a challenge. This allowed to vary the response size, without additional hardware. The experiment was tested on an Artix-7 Nexys 4 FPGA board. The results in uniqueness of the bits of 49.2%. The final response, after the consecutive PUF's challenge response stream, showed an accuracy of 99.99 %.

In [33], Xu et al. proposed a RFID authentication protocol based on Physical Unclonable Functions. The PUF module is embedded within the RFID module. Hence, each RFID tag will have a self-generate a unique key based on its own embedded circuit. The tag and the reader exchange the secret key for verification. This key is periodically changed. The protocol was tested using standard methods to test the protocol such as GNY [34], where protocol was tested to persist against different type of attacks such as tag forgery, steak attack, replay attack, backtracking attack, and clone attack. This was also experimented using a set up involving a simulated cargo vehicle with Ultrahigh frequency RFID. The success rate of the protocol implementation was 91.90%.

In [35], Jaafar proposed a similar approach was taken but device authentication was performed using Cross-layer Response Times, that is, how quickly a device was responding to different queries. These were then compared via the expected results stored earlier. In [36], the research proposed a method that leveraged Near Field Communication (NFC) as a fingerprint for a smartphone, to authenticate a user into a library system. The user's identity is generated from the fingerprint associated with the NFC in their smartphone.

Other research leveraged the on-device equipment's to perform fingerprinting. A typical illustration of this approach is presented in [37]. Here, the fingerprints were generated using the audio connectors and accelerometers on the IoT devices. The frequency response of the speakerphone-microphone system, and device-specific accelerometer calibration errors were also leveraged. The features used were mean, standard deviation, average deviation, skewness, kurtosis, RMS amplitude, minimum, and maximum of every chunk of accelerometer data.

The approach of comparing various fingerprints based on hardware or networking features with an existing database even along with the error tolerated through using distances similar to the hamming distance does not generalize to an IoT environment. This is because a typical IoT environment involves devices entering and leaving the network often, and on typically on a large scale. This means that these classical approaches where fingerprints are pre-calculated would have limited applicability given the variety and large number of devices.

In [38], B. Charyyev et al. developed a Locality-Sensitive IoT Fingerprinting (LSIF) that relies on a Locality Sensitive Hash of the traffic flow to identify a device. The data consisted of 3 datasets, from 111 different smart home IoT devices collected over a period of 20 days. Each dataset consisted of data of devices at different states, namely, setup of device, idle, active state. The hash generated from the traffic flow was used as a fingerprint to distinguish between the devices. The reported result shows a precision of 0.83 when in setup state, 0.94 when in idle state, and 0.94 when in active state too.

**3.2.2. Fingerprinting using machine learning.** In this section, the work done in the literature for fingerprinting that leverage machine learning methods and artificial neural networks are discussed. This is divided into 3 sections, autoencoders, supervised and unsupervised learning. The literature work that leverages autoencoders are discussed in section 3.2.2.1. The work for the supervised learning algorithms is discussed in section 3.2.2.2, and the ones that leverage unsupervised learning algorithms are discussed in 3.2.2.3.

**3.2.2.1. Fingerprinting using autoencoders.** One approach to such a problem is by creating a fingerprint for every IoT edge device that uniquely identifies it. These fingerprints be created based on various types of device features. One type of fingerprints can be generated by using a type of Artificial Neural Network (ANN) called autoencoders. Autoencoders are used to generate efficient codes for inputs in an unsupervised manner. This means that every device will have a unique code, called a fingerprint, that will be used to uniquely identify it. This layer is employed as the pre-trained layers of the Neural Network. The fingerprint will be used by the classical machine learning models to identify the device. In this section, we describe what are autoencoders and how they are leveraged to reduce the dimensionality of large scale IoT edge device identification problem.

An autoencoder is an Artificial Neural Network model that produces an output that is similar to the output. In other words, autoencoders can reconstruct an output from a given input [39] and are presented in Figure 3.1 .

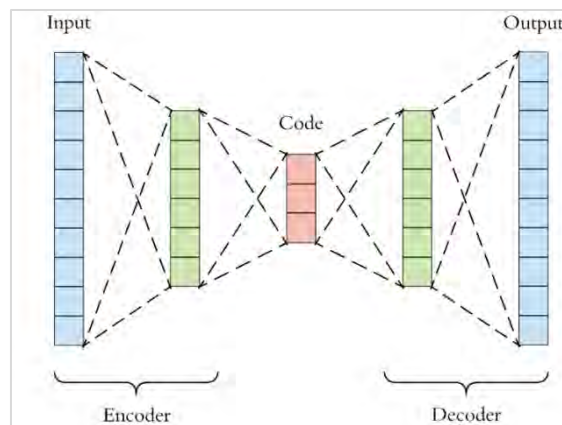


Figure 3.1: Autoencoders [40].

Autoencoders have not been extensively used in the research for addressing the problem of device identification. In [41-43], autoencoders were used for content-based image retrieval. The methods generalize to tuning an autoencoder for dimensionality reduction followed by a classification algorithm. In [44], the data used consisted of 1.6 million images, where the data were partially labelled only. The approach taken in constructing the autoencoder was a 28-bit and a 256-bit autoencoder, their weights from two separately trained stacks of Restricted Boltzmann Machines (RNN). In each autoencoder, the hidden layers halve in size until they reach the desired size.

In [45], Soren used autoencoders to classify 36,000 images. The resulting model was tested using new images or altered images from the training set. The autoencoder were able to successfully reconstruct and classify these images. Two classification tests were done, correct image and correct image in the correct set. The results of accuracies were 97% and 91% respectively.

Autoencoders can be a good candidate for properly addressing the large number of classes, the unseen new classes challenge in the context of IoT. Therefore, it is a promising technique when combining them with the traditional Neural Networks model for the identification problem.

**3.2.2.2. Supervised learning methods.** Random Forest, Naïve Bayes, and Gradient Boosting [46-48] were the most commonly used methods used. For example, in [49], Random Forest and Gradient Boosting algorithms were used to classify different devices from darknet using information from IP headers as the feature set. These two techniques were empirically experimented with for about 3 million devices. The classification using both methods resulted in 93.9% and 99.8% recall results, and 93.7% and 99.8% precision values for Random Forest and Gradient Boost respectively.

The precision values were [50] also showed the success of Random Forest for this problem. The random forest was evaluated attack using traffic traces collected using tor browsers nodes. Accuracy was almost 99.98% when number of trees were 6 and above. In [51], features extracted from the TCP/IP traffic were fed into Random Forest model to classify 17 different IoT smart home devices such as a baby monitor, smoke detector, TV, refrigerator etc. Finally, [52] leveraged SVM to leverage minute imperfections of IEEE 802.11 transmissions to uniquely identify IoT devices. This algorithm is typically used when a device has to be classified into a certain class.

In [53], Hameed et al. proposed a classification framework that utilizes both the network and statistical features to characterize the IoT devices in the context of a smart city. It is to identify the device falls under which class. The dataset used is a publicly available data set, containing data been collected over a month for 21 different IoT devices, which include devices like Amazon Echo, Samsung SmartCam, Iphone, MacBook etc. The dataset is consisting of 101,922 labelled instances were collected from the traffic traces of these 21 devices. Features from network were leveraged to train

the model and are source IP, destination IP, IP protocol, port number, destination port number, TTL, Ethernet address, Inter-Arrival time, average packet size, and traffic rate. Four types of models were built: Random Forest, Decision Tree, SVM and Gradient Boosting. The models were trained using 70% training and 30 % testing split. The reported goodness of measure values is shown in Table 3.1. It can be shown that Gradient Boosting outperformed other models.

Table 3.1: Models Result in [53].

Model	Precision	Recall	F1 Score
<b>Random Forest</b>	0.8	0.78	0.77
<b>Decision Tree</b>	0.84	0.766	0.8
<b>SVM</b>	0.85	0.76	0.84
<b>Gradient Boosting</b>	0.9	1	0.94

In [54], Hussain et al. proposed a machine learning-based attack that exploits network patterns to identify smart IoT devices and running services. The setup used to collect data consisted of a smart home with three smart home devices: Amazon Echo, Amazon Echo Dot, and Google Nest Mini. The data has been collected over 8 hours. The features leveraged are the Interarrival time and the Packet size. This input features were converted to the statistical features: standard deviation, sum, variance, maximum, minimum, mean, median, skewness, and kurtosis. A Random Forest model was trained for it. The results of the trained model are shown in Table 3.2.

Table 3.2: Accuracy of Model per Service in [54].

Model	Service	Accuracy
<b>Amazon Echo Dot</b>	Music	0.9992
	News	0.9994
<b>Amazon Echo</b>	News	0.9996
	Music	0.9997
<b>Google Nest Mini</b>	Music	0.9997

In [55], Hussain et al. leverages six machine learning models, Random Forest, Decision Tree, SVM, Nearest Neighbors, Gaussian Naïve Bayes and an Artificial

Neural Network to distinguish between devices in a smart environment. The environment used to generate the data consists of Smart home containing 4 types of IoT devices connected. These devices are TP-Link Connected Bulb, Nest Security Camera, Mini, D-Link Motion Detector, and Wemo Switch Smart Plug. By analysing streams of packets sent and received, machine learning leveraged to identify which IoT device generated which traffic. This network data is basically the network traffic that has been collected for 7 days. There was a total of 3,222 flows for training and 805 flows for testing. The features leveraged are the size of the first N packets sent, the size of the first N packets received, the N - 1 packet inter-arrival times between the first N packets sent, and the N - 1 packet inter-arrival times between the first N packets received. The results in terms of the precision and recall are shown in Table 3.3. In their experiment, Random Forest model outperformed all others.

Table 3.3: Models Result in [55].

Model	Precision	Recall
<b>Random Forest</b>	0.999	0.999
<b>Decision Tree</b>	0.995	0.995
<b>SVM</b>	0.993	0.993
<b>K-NN</b>	0.989	0.989
<b>Gradient Boosting</b>	0.919	0.919
<b>Artificial Neural Networks – Fully Connected Feed Forward</b>	0.986	0.986

In [56] Saciancalepore et al aimed to create a fingerprint for drones in order to detect the presence of a drone, and its current status. This was performed by training a Random Forest model. The data collected was traces of traffic originated by a drone, that had 2 different states: steady state and movement state. The size of the collected packets was 174,677. The features leveraged to the train the model were Interarrival time and the packet size. The result reported in the work is an accuracy of 0.97.

In [57] Chaddad et al. proposed a method for the identification of mobile applications by training four models: Random Forest, Bagged Tree, SVM, and K-NN.

The data consists of descriptions of application traffic flows for six applications running on Android mobile phones. The traffic flow was extracted for every one second, and the leveraged feature for training the model was Inter-Arrival time. The precision results of the models are shown in Table 3.4.

Table 3.4: Precision Results for the Models in [57].

Model	Precision
<b>Random Forest</b>	0.911
<b>Bagged Tree</b>	0.90
<b>SVM</b>	0.767
<b>K-NN</b>	0.839

In [58], Babun et al. proposed a Z-IoT fingerprinting framework aimed to distinguish between 39 of IoT classes of device classes. These devices leverage the ZigBee and Z-Wave protocols for communication. The Z-IoT framework monitors the idle network traffic to implement device-class fingerprinting that are based on signatures. The leveraged features used to train the model is the Inter-Arrival time, and the model is to create a signature (density distribution, splits into bins and area under each bin). The criteria for selection are the best ML-Filter settings approach to select the best model results. The results of the accuracy and precision is shown in Table 3.5 for each of the device types.

Table 3.5: Accuracy and Precision Results for the Models in [58].

Device Type	Accuracy	Precision
<b>ZigBee</b>	0.91	0.912
<b>Z-Wave</b>	0.92	0.936

In [59], Vaidya et al. proposed is a model that will identify between 50 IoT devices by leveraging the device characteristics (Physically Unclonable Functions) to create device fingerprints. The setup consisted of 50 IoT devices, running over a period of one month, where data has been captured every minute, resulting in a total number of samples of 500,000 samples. The leveraged features were Clockcount, ADCsingle, ADCdiff and a Bonsai model was used. Bonsai model is a machine learning model

based on decision trees. The training and testing data was split into 50% each. The reported accuracy results are shown in Table 3.6.

Table 3.6: Accuracy of Bonsai Model in [59].

Feature	Accuracy
<b>Clock count</b>	0.54
<b>ADC (single + diff)</b>	0.872
<b>Combined</b>	1

In [60], V. Rimmer et al. used Deep Learning (DL), Convolutional Neural Networks (CNN) and Recurrent Long-Short Term Memory (LSTM) methods to perform the classification. The IoT data used was generated from traffic features generated from website fingerprinting. The data for training consisted of 900 websites, with 400,000 traffic test traces. The models were tested using new unknown websites in the existing dataset. The accuracy results of the models are shown in Table 3.7.

Table 3.7: Accuracy of Models in [60].

Model	Accuracy
<b>CNN</b>	0.96
<b>Deep Learning</b>	0.95
<b>LSTM</b>	0.94

Similarly, Sun et al. in [61] used DL to classify 10,000 classes of IoT devices. Their approach used four convolutional layers to automatically extract the features from the network traffic. The hidden neurons were ReLUs and the output neuron was a sigmoid A Softmax layer followed to identify the final class. For verification, the Neural Network contains one input layer, one locally connected layer, one fully connected layer, and a single output neuron indicating the similarities. The resultant accuracy obtained was 97.45%.

Qiu et al. in [62] leveraged CNN to detect network spoofing attacks for three device types. The three device types are: a smart device, that is capable of performing advanced tasks, devices that resources limited, and devices trying to send misleading message to the smart device. This was performed by developing a lightweight scheme

that is based on Neural Network classifier. CNN was leveraged to learn the features from the input, and the work has been compared against the traditional machine learning models, SVM, Gaussian Mixture Model (GMM) and against a Physical Layer Authentication (PLA) model. The accuracy (detection rate) of these models is shown in Table 3.8. It can be shown that CNN and GMM performed the same and outperformed SVM.

Table 3.8: Model Results in [62].

Model	SVM	GMM	CNN
<b>Detection Rate</b>	0.91	0.95	0.95

Spanos et al. in [63] addressed a larger number of instances of IoT devices connected to the Internet. About 15.3 million devices were classified using a Neural Network that leverages Long Short-Term Memory. The data was collected using Amazon Web crawlers and included 41 device types, 1664 vendors, and 12,800 products. The classification was performed at three levels: device-type, vendor-type and product-type. Features were extracted from the application layer, the network layer, and the transport layer. Some of these features for network and transport layer include, Recovery Time Objective, Time to Live, type of service. For application layer, the feature analysis was performed on 20 protocols, few were TCP, SSH, Telnet. The Neural Network generated IoT device fingerprints in a three-level breakdown: type, vendor, and product. Results showed good performance with 94% precision and 95% recall.

In [64], Skowron et al. proposed a CNN that have low computational complexity and is used to uniquely identify between the devices. The environment setup consisted of 9 smart home IoT devices, where their network traffic was captured. The extracted network features that were leveraged to train the model are packet sizes, time to live, etc. The CNN built consisted of 2 consecutive Convolutional layers, followed by a Dense (256) layer, a Dense (128) layer, a Dense (64) layer, and a Softmax layer. The CNN model was then compared with the traditional machine learning models are SVM, Random Forest, and Decision Trees. These results are shown in Table 3.9. From the results, it can interpret that CNN outperformed other models in identification between the devices.

Table 3.9: CNN and other Model Results in [64].

Model	Accuracy	Precision	Recall	F1-Score
<b>SVM</b>	0.81	0.78	0.81	0.77
<b>Random Forest</b>	0.61	0.67	0.62	0.8
<b>Decision Trees</b>	0.57	0.53	0.48	0.84
<b>CNN</b>	0.87	0.92	0.88	0.94

In [65], Kotak et al. leveraged CNN to distinguish between 10 different IoT devices such as a baby monitor, lights, motion sensors, security cameras, smoke detectors etc. A model was built for each class of devices. The dataset used for these devices is a public dataset and consisted of 218,657 TCP sessions. The features used to train the model are Source IP, Destination IP, Port number, and Protocol. The model consisted of an input layer and an output layer. The input to the model is a  $28 \times 28$  pixel grayscale image. Each of the input was first converted to a binary type before the model was trained. The reported accuracy of the models was around 0.99.

In [66], Aneja et al. proposed a method to leverage CNN to identify between 2 classes of devices. These devices were iPad and an iPhone. The setup involved collecting network traffic from these two devices. The feature leveraged from the traffic is the Inter-Arrival time. A CNN model was trained using this data and had an architecture of Rectified Linear Unit (ReLU), 2D Max Pooling layer, two Convolution and Max Pooling layers, with batch size of 16. The reported accuracy of the model was 0.86.

In [67], Simpson et al. built 2 models, CNN and K-NN to classify the flows of traffic from network devices. The work aimed to perform this identification to look for ambiguous flow in the network traffic. The dataset consisted of 4,994 TCP flows, with each flow being around 3 seconds. The leveraged feature was the Inter-Arrival time and was used to train both models. The CNN architecture consisted of Convolution and Max Pooling layers, followed by another Convolution and Max Pooling layers and a Dense Layer. The reported CNN F1-score was 0.965 and KNN F1-score was 0.864, showing that CNN outperformed the K-NN model to perform such a classification.

Salman et al. in [68] leverages CNN to classify network traffic for a set of 6 classes IoT devices. These IoT devices were D-Link Water Sensor, D-Link Camera, D-

Link Siren, D-Link Plug, and Samsung Home Kit. The data collected was performed one hour for each of the devices. The features leveraged for training was the Inter-Arrival time, protocol, and direction. The CNN architecture consisted of 2 Convolutional layers, 2 Pooling layers, and 2 Fully-Connected layers. The data has been represented in form of an image with size (28 by 28), with both a gray scale and a coloured image. The reported F1-Score for gray representation was 0.78, and for the coloured representation was 0.86. The model was compared with Random Forest, and the reported F1-Score for Random Forest was 0.93 (outperforming the CNN models).

**3.2.2.3. Unsupervised learning methods.** In [69], Thangavelu et al. K-means clustering was used with features are extracted from network traffic of smart home devices using features shown in Table 3.10. From the features extracted, the model is train to group the features into different clusters. A threshold is set to compare the mean and standard deviation scores of the data points, and accordingly a decision for it to fall on which cluster is made. Clustering results showed high performance with F-score, Precision, and Recall all above 0.9. While the traditional machine learning methods could accurately discriminate among these devices, they might not generalize to new devices.

Table 3.10: List of Fingerprinting Features in [69].

Protocol	Feature
<b>DNS</b>	DNS queries, DNS errors, DNS packet count, Mode of domain name queries, DNS packet length, DNS query response time
<b>Session statistics</b>	Number of packets in session, active duration
<b>TLS</b>	Number of TCP keep a live, TLS packet count, length, TLS flow
<b>HTTP</b>	Number of TTP response code, method, #TCP keep alive, HTTP request URI, HTTP packet count, length, HTTP flow duration
<b>SSDP</b>	SSDP packet count, length, SSDP flow duration
<b>QUIC</b>	QUIC packet count, length, QUIC flow duration, QUIC packet type
<b>MQTT</b>	MQTT packet count, length, MQTT flow duration, MQTT packet type
<b>STUN</b>	STUN packet count, length, STUN flow duration
<b>NTP</b>	NTP packet count, length, NTP flow duration
<b>BOOTP</b>	BOOTP packet count, length, BOOTP flow duration

Merchant et al. in [70], CNNs were used to detect the attributes at the physical layer for the identification of radio devices on some IEEE 802.15.4 devices. Data was collected from seven ZigBee devices. For each device, timeslots of seconds of data were digitized to train a CNN so that device identity could be derived based upon device-specific imperfections in these transmissions. The problem was converted into binary pair-wise classification problem where one device was positive, and all other rest were negative exemplars. The identification accuracy was around 92%.

In [71], Qiu et al. proposed an anomaly detection mechanism that leverage network traffic from 9 different IoT devices, which include a weigh scale, blood pressure meter, emergency button, and motion sensors. These 9 devices run different protocols such as Bluetooth, the RF869, and the ZigBee. The features extracted from the network are size of the packets, number of packets, number of event packets, event, duration. The leveraged machine learning model is DBSCAN [72]. The accuracy, recall, and F1-score of the model is shown in Table 3.11.

Table 3.11: Results of DBSCAN Model in [71].

Measure	Accuracy	Recall	F-score
Value	0.861	0.833	0.90

In [73] Qing et al., proposed a method that leverages CNN to produce a Radio Frequency Fingerprinting (RFF) fingerprints to distinguish between 54 IoT devices. These IoT devices run ZigBee, and the ZigBee Radio Frequency Waveforms have been collected from these devices. The features have been extracted by a CNN, that consisted of 3 Convolutional layers, followed by a Softmax layer. A Differential Constellation Trace Figure (DCTF) is utilized to extract the RFF features. The model resulted in an accuracy of 0.99.

A summary of the literature for authentication using machine learning is shown in Tables 3.12 to Table 3.15. It presents the data or device type, whether the devices were identical, the number of devices, the features leveraged to train the model, the models trained including the best model. It can be shown that the work in the literature address the identification between devices that are not identical (different device types) and not identical ones.

Table 3.12: Summary of Authentication using Machine Learning – Part 1.

Ref.	Device / Data type	Identical Devices	No. of Devices (if applicable)	Features used	Models Trained	Result
[46]	10 Internet applications, 40 traces for each	Not provided	Not provided	IAT, packet size, burst size,	Naïve Bayes	Acc: 0.94
					Regression	Acc: 0.95
					Random Forest (Best model)	Acc: 0.96
[48]	49 Android and iOS applications	Not provided	Not provided	Statistical features of incoming and outgoing packets of applications	Random Forest	F1: 0.72
[49]	3M Devices (darknet)	No	3M Devices	IP headers data	Random Forest	Prec: 0.93
					Gradient Boost	Prec: 0.99
[50]	722 hidden services from the Internet	Not provided	Not provided	Not mentioned	Random Forest	Prec: 0.98
[51]	Baby monitor, smoke detector, TV, refrigerator etc.	No	17	TCP/IP features from packets	Random Forest	Acc: 0.98
[52]	130 identical 802.11 NICs	Yes	130	SYNC correlation, I/Q offset, magnitude / phase errors; SYNC correlation, I/Q offset	SVM	Acc: 0.9
[53]	Amazon Echo, Samsung SmartCam, Iphone, MacBook etc.	No	21	Source IP, destination IP, IP protocol, port number, destination port number, TTL, Ethernet address, IAT, average packet size, traffic rate	Random Forest	F1: 0.77
					Decision Tree	F1: 0.8
					SVM	F1: 0.84
					Gradient Boosting (Best model)	F1: 0.94

Table 3.13: Summary of Authentication using Machine Learning – Part 2.

Ref .	Device / Data type	Identical Devices	No. of Devices (if applicable)	Features used	Models Trained	Result
[54]	Amazon Echo, Amazon Echo Dot, and Google Nest Mini	No	3	IAT, packet size	Random Forest	Accuracy 0.99
[55]	Link Connected Bulb, Nest Security Camera, Mini, D-Link Motion Detector, Wemo Switch Smart Plug	No	4	Size of the first N packets sent, size of the first N packets received, N – 1 packet inter-arrival times between the first N packets sent, and the N – 1 packet inter-arrival times between the first N packets received.	Random Forest <b>(Best Model)</b>	Prec: 0.999
					Decision Tree	Prec: 0.995
					SVM	Prec: 0.993
					K-NN	Prec: 0.989
					Gradient Boost	Prec: 0.919
					ANN (Fully Connected)	Prec: 0.986
[56]	Drones (collected packets was 174,677)	Yes	Not provided	IAT, packet size	Random Forest	Acc: 0.97
[57]	6 mobile applications	Not provided	Not provided	IAT	Random Forest <b>(Best model)</b>	Prec: 0.911
					Bagged Tree	Prec: 0.90
					SVM	Prec: 0.767
					K-NN	Prec: 0.839
[58]	ZigBee and Z-Wave IoT devices	No	39	IAT	Best ML-Filter settings approach	Prec: 0.93
[59]	Device type not mentioned	No	50	Clockcount, ADCsingle, ADCdiff	Bonsai (DT)	Acc: 0.87

Table 3.14: Summary of Authentication using Machine Learning – Part 3.

Ref.	Device / Data type	Identical Devices	No. of Devices (if applicable)	Features used	Models Trained	Result
[60]	900 websites, with 400,000 traffic test traces	Not provided	Not provided	Learnt by model	CNN <b>(Best model)</b>	Acc: 0.96
					LSTM	Acc: 0.94
[61]	10,000 classes of IoT devices	No	10,000	Learnt by model	CNN	Acc: 0.97
[62]	The three device types are: a smart device, that is capable of performing advanced tasks, devices that resources limited	No	3	Learnt by model	SVM	Det. Rate: 0.91
					Gaussian Mixture Model (GMM)	Det. Rate: 0.95
					CNN	Det. Rate: 0.95
[63]	41 device types, 1664 vendors, and 12,800 products	No	15.3M	Recovery Time Objective, Time to Live, type of service	LSTM	Prec: 0.94
[64]	Smart home IoT devices	No	9	Packet sizes, time to live	SVM	F1: 0.77
					Random Forest	F1: 0.8
					Decision Tree	F1: 0.84
					CNN <b>(best model)</b>	F1: 0.94
[65]	Baby monitor, lights, motion sensors, security cameras, smoke detectors etc	No	10	Source IP, Destination IP, Port number, and Protocol	CNN	Acc: 0.99
[66]	iPad and an iPhone	No	2	IAT	CNN	Acc: 0.86

Table 3.15: Summary of Authentication using Machine Learning - Cont'd.

Ref.	Device / Data type	Identical Devices	No. of Devices (if applicable)	Features used	Models Trained	Result
[67]	4,994 TCP flows	Not provided	Not provided	IAT	CNN	F1:0.96
					K-NN	F1:0.86
[68]	IoT devices were D-Link Water Sensor, D-Link Camera, D-Link Siren, D-Link Plug, and Samsung Home Kit	No	6	Inter-Arrival time, protocol, and direction	CNN	F1:0.86
					Random Forest	F1:0.93
[69]	Smart home devices	No	Not mentioned	Features are extracted from network traffic (DNS, TLS, HTTP) etc.	K-Means	F1:0.9
[70]	ZigBee devices	No	7	Learnt by model	CNN	Acc:0.92
[71]	Weigh scale, blood pressure meter, emergency button, and motion sensors, etc	No	9	Size of the packets, number of packets, number of event packets, event, duration	DBSCAN	F1:0.90
[72]	Devices run ZigBee, and the ZigBee Radio Frequency Waveforms	No	54	Learnt by model	CNN	Acc:0.99

## Chapter 4. Methodology

In this chapter, the method followed in order to setup the experiment of the thesis and how the work will be assessed described. This is divided in five sections. In section 4.1, the use case scenario followed to build the set up and generate the data is explained. Following that in section 4.2, the data generated from the experiment is described. In section 4.3, the processing that was performed on the data that will be used for training the model is explained. In section 4.4, the models that were built, including the traditional models and the Convolutional Neural Networks and how they were trained is described. Finally, in section 4.5, the evaluation criteria performed on the results to check for the accuracy of the results is presented.

### 4.1. Use Case Scenario

The use case scenario is to simulate a smart home with edge devices deployed for the different smart home devices. These edge devices are simulated using ESP32 devices that will communicate with each other and with the server via MQTT protocol [74]. There are 18 of these ESP32 devices. The code is run to simulate a smart home device that will switch on and off during the day for different purposes. This is described.

#### **Scenario:**

An example of the scenario is a surveillance camera in a smart home environment. The camera is to switch on when the homeowners are outside the home, and off when they are in. The sequence of these operations throughout the day is as follows.

- The smart device will need to be off when the homeowners are present at home and will need to be on when homeowners are away or in the nighttime. The status of the device is to be checked every second too.
- It is assumed that homeowners will be away from 9 AM onwards.
- It is assumed that homeowners will be present from 5 PM onwards.
- From 12 AM to 6 AM is nighttime, and therefore the smart device must be on.
- A heartbeat (status) message is sent every 1 second to check the liveliness of the device.

This is presented in Figure 4.1:

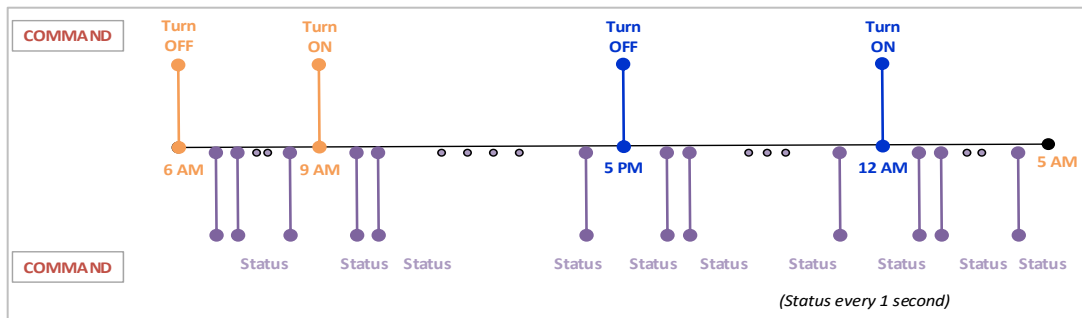


Figure 4.1: Use Case Scenario for a Surveillance Camera in a Smart Home

## 4.2. Data Description

The data generated were from 18 ESP32 devices, that ran and communicated via the MQTT protocol. The data consisted of TCP and MQTT packets, for devices communicating over a period of approximately 28 days. The data was captured via Wireshark that captured the communicating packets and stores them into a pcap file [75]. The network, transmission, and MQTT features, and their sizes are shown in Table 4.1, Table 4.2, and Table 4.3. Table 4.1

Table 4.1: Network Layer Features.

Features	Data type	Size in packet
Packet type	String	2 bytes
Link-layer address type	Numerical	2 bytes
Link-layer address length	Numerical	2 bytes
Protocol	Categorical	2 bytes
Header length	Numerical	2 bytes
Total length	Numerical	2 bytes
Identification	String	2 bytes
Reserved Flag	Categorical	1 bit
Don't Fragment Flag	Categorical	1 bit
More Fragment Flag	Categorical	1 bit
Fragment Offset Flag	Categorical	1 bit
Time to Live	Numerical	1 byte
Protocol	String	1 byte
Source IP	Numerical	4 bytes
Destination IP	Numerical	4 bytes

Table 4.2: Transmission Layer Features.

Features	Data type	Size in packet
<b>Src Port</b>	Numerical	2 bytes
<b>Dst Port</b>	Numerical	2 bytes
<b>Sequence Number</b>	Numerical	4 bytes
<b>Ack</b>	Categorical	1 bit
<b>Ack Number</b>	Numerical	4 bytes
<b>Header length</b>	Numerical	20 bytes
<b>Flags: Reserved, Nonce, Congestion Window Reduced, ECN-Echo, Urgent, Acknowledgment, Push, Reset, Syn, Fin</b>	Categorical	2 bytes
<b>Window size value</b>	Numerical	2 bytes
<b>Bytes in flight</b>	Numerical	4 bytes
<b>TCP Payload size</b>	Numerical	Variable
<b>PDU size</b>	Numerical	Variable

Table 4.3: MQTT Features.

Features	Data type	Size in packet
<b>Message Type</b>	Categorical	1 byte
<b>DUP Flag</b>	Categorical	1 bit
<b>QoS level</b>	Categorical	1 bit
<b>Retain</b>	Categorical	1 bit
<b>Msg Length</b>	Numerical	1 byte
<b>Topic Length</b>	Numerical	2 bytes
<b>Topic</b>	String	Variable
<b>Message</b>	String	Variable
<b>Roundtrip Time</b>	Numerical	1 byte
<b>Control Message</b>	Categorical	3 bits
<b>ProtocolName</b>	Categorical	3 bits
<b>ProtocolLevel</b>	Categorical	1 bit
<b>Connect Flags</b>	Categorical	1 bit
<b>Keep Alive</b>	Categorical	2 bits
<b>Payload</b>	Numerical	Variable

For the 18 devices, the summary of the data is shown in Table 4.4.

Table 4.4: Data Summary.

Data Description	Value
Number of Devices	18
Number of days of Data Collection	28
Device Type	ESP32
Communication Protocol	MQTT (Over TCP)
Communication Model	Publish / Subscribe
Number of Data Points Per Device	~ 100,400
Missing Data	N/A
Balanced Data	Yes
Selected Feature	Inter-Arrival Time

### 4.3. Data Preprocessing

The data generated was from 18 ESP32 devices. Since the devices were communicating and exchanging messages via MQTT protocol, the generated data contains network, transmission data (TCP and MQTT sessions). In this thesis, we study and build experiments for three types of experiments:

- Traditional Machine Learning Models (such as Random Forest, Bayesian, etc.) with Inter-Arrival Time feature
- Convolutional Neural Networks with Inter-Arrival Time feature
- Convolutional Neural Networks with binary input

#### **Traditional Machine Learning Models:**

1. First, the Inter-Arrival time has been captured from the data generated. Since the data has been captured using Wireshark [76], one can get the time between the packets from the pcaps file themselves. In this way, for every device, we have the time between every 2 packets in separate files.
2. Next, the data has been sampled, using sampling frequency using a sampling rate of 10,000 Hz.

3. Finally, the statistical information about the Inter-Arrival time data was calculated.

Eight statistical features are calculated which were:

- Mean
- Maximum
- Minimum
- Variance
- Kurtosis
- Skewness
- Median
- Standard Deviation

This data was then normalized and used to train the model. A capture from this data is shown in Figure 4.2.

	Mean	Max	Min	Var	Kurtosis	Skewness	Median
26720	0.319218	0.059579	0.0	0.027396	0.005019	0.038071	0.0
35807	0.332858	0.229562	0.0	0.145189	0.073203	0.265051	0.0
8791	0.461584	0.063499	0.0	0.033448	0.004458	0.051902	0.0
56231	0.296990	0.062084	0.0	0.025075	0.003465	0.041045	0.0
36591	0.339456	0.046892	0.0	0.028128	0.004527	0.051246	0.0
...	...	...	...	...	...	...	...
53724	0.358295	0.062831	0.0	0.028800	0.002481	0.042078	0.0
19099	0.336861	0.509333	0.0	0.308891	0.154719	0.387078	0.0
43167	0.366564	0.059661	0.0	0.028820	0.004422	0.049837	0.0
76164	0.439057	0.040290	0.0	0.036679	0.004609	0.061127	0.0
6571	0.405885	0.063335	0.0	0.031908	0.004453	0.051743	0.0
	Std	Device					
26720	0.205140	5					
35807	0.381035	7					
8791	0.605905	1					
56231	0.336995	11					
36591	0.325767	7					
...	...	...					
53724	0.375291	10					
19099	0.555777	3					
43167	0.355063	8					
76164	0.571342	15					
6571	0.603135	1					

Figure 4.2: Statistical Features of Data.

### **Convolutional Neural Networks with Inter-Arrival Time feature:**

For CNN, the data has been converted to the LogMel [77] values before training the CNN model. A detailed explanation of this is present in section 7.2. This is performed as follows [78]:

1. First, the Inter-Arrival time has been captured from the data generated. Since the data has been captured using Wireshark, one can get the time between the packets from the pcaps file themselves. In this way, for every device, we have the time between every 2 packets in separate files.
2. Next, the data has been sampled, using sampling frequency using a sampling rate of 10,000 Hz.
3. Framing the signal. This involves framing our generated data (IAT) into very small windows of **50 milliseconds** each.
4. **Fast Fourier Transform** is applied to each of the frames, to provide the spectral components and frequency information in each frame.
5. **Periodogram** is calculated for each of the resultant frames, to identify which frequencies are present in the frame.
6. Compute the **Mel Filter Banks**.
7. **Apply** the Mel Filter Banks to the Periodograms.
8. **Sum up** the above results to get an idea of how much energy exists in various frequency regions. Take the **log** of the above result. The result of this is scaled and used as an input to the CNN model. A capture from this data is shown in Figure 4.3.

		1	2	3	4	5	6	7	
0		0.749850	0.772100	0.791730	0.791730	0.791730	0.825180	0.839680	
1		0.787770	0.788700	0.841870	0.836190	0.824860	0.857660	0.877750	
2		0.809550	0.792430	0.923640	0.886360	0.816370	0.942820	0.913640	
3		0.764610	0.819910	0.864400	0.815020	0.864640	0.867210	0.897940	
4		0.833570	0.826510	0.930180	0.884190	0.855770	0.961990	0.908250	
...		...	...	...	...	...	...	...	
56899		0.822562	0.830022	0.861572	0.849052	0.869242	0.895922	0.907712	
56900		0.718642	0.731752	0.745352	0.754512	0.758452	0.778022	0.802922	
56901		0.880502	0.873342	0.954392	0.929142	0.901882	0.992932	0.958042	
56902		0.819452	0.857592	0.863032	0.876132	0.862922	0.903672	0.915502	
56903		0.868852	0.903702	0.894642	0.881452	0.868962	0.868362	0.856932	
		8	9	10	...	251	252	253	\
0		0.853020	0.865370	0.876870	...	0.907260	0.873570	0.921540	
1		0.886280	0.906170	0.917860	...	0.721290	0.715120	0.750370	
2		0.942130	0.951110	0.968900	...	0.812170	0.809930	0.838210	
3		0.897720	0.929010	0.919440	...	0.836780	0.839630	0.866760	
4		0.974930	0.947220	0.991620	...	0.739840	0.727500	0.768240	
...		...	...	...	...	...	...	...	
56899		0.915962	0.931362	0.950692	...	0.928182	0.867232	1.002822	
56900		0.812952	0.825582	0.836742	...	0.705002	0.720552	0.744732	
56901		0.998432	0.995682	1.003112	...	0.957402	0.821892	0.802912	
56902		0.929532	0.944382	0.953582	...	0.695772	0.698902	0.729002	
56903		0.841612	0.851502	0.888532	...	0.671752	0.658292	0.702392	
		254	255	256	257	258	259	device	
0		0.931800	0.941820	0.964290	0.974770	0.991230	1.000000	1	
1		0.769040	0.777910	0.791580	0.804380	0.823190	0.834290	1	
2		0.839940	0.879730	0.879700	0.881740	0.914720	0.918740	1	
3		0.883970	0.899260	0.907710	0.923060	0.941200	0.952900	1	
4		0.787050	0.787060	0.814390	0.815270	0.840140	0.849310	1	
...		...	...	...	...	...	...	...	
56899		0.907162	1.003112	0.969492	1.003112	1.003112	1.003112	1	
56900		0.761782	0.776042	0.784722	0.802582	0.818122	0.829332	1	
56901		0.978672	0.890572	0.927082	0.983402	0.934742	1.001552	1	
56902		0.739992	0.765622	0.763252	0.787692	0.799982	0.811642	1	
56903		0.713712	0.725202	0.740992	0.749552	0.772992	0.781262	1	

Figure 4.3: Mel Values of Data.

### Convolutional Neural Networks with binary input:

The following processing to the input was performed:

1. From the captured data files (pcaps) of the 18 ESP32 devices, the files were grouped based on their Source and Destination IP address, resulting in each pcap file containing the sessions from one unique device only.
2. One second data from each of the files were captured. This was done by looking at the time of between the packets. For each approximate of one second, the associated packets in that frame are separated.
3. Since each one second is approximately 1000 bytes (some are less, some are more), and in order to be able to convert it to 28 by 28 input, each one second bytes of data were converted to 784 bytes. This was done by either trimming the bytes or padding them with 0 till they reach 784 bytes.
4. Each of these 784 bytes were converted to binary data.
5. This was performed and gathered for each of the 18 devices, and the 28 by 28 bytes were used as an input to the model.

#### 4.4. Model Building

There are three types of experiments in this thesis. The first one leverages traditional machine learning models. The second and third builds a CNN models with different inputs. This is summarized in Table 4.5.

Table 4.5: Experiment Types.

Experiment	Model Name
<b>Experiment 1:</b>	Random Forest
	Bayesian
	Support Vector Machine
	Light Gradient Boosting Machine(LightGBM)
	Gradient Boost
<b>Experiment 2:</b>	Convolutional Neural Networks with log mel input
<b>Experiment 3:</b>	Convolutional Neural Networks with binary input

#### 4.5. Models Evaluation Criteria

In this section, how the models were evaluated is presented. These are measures performed to test accurate the model results are.

**4.5.1. Confusion matrix.** In classification problems, there are four outcomes that will occur [79]:

- **True positive:** is when an observation that is predicted belongs to a class and in fact it does belong to that class (also called Sensitivity).
- **True negative:** is when an observation that is predicted does not belong to a class and it actually does not belong to that class.
- **False positive:** is when an observation that is predicted belongs to a class when in in fact it does not (also called Specificity).
- **False negative:** is when an observation that is predicted does not belong to a class when in reality it does.

These four outcomes are often plotted on a confusion matrix [80]. This is shown in Figure 4.4. In classification problems, confusion matrix is used to provide a breakdown with the number of correct and incorrect predictions or each of the classes. This can indicate how well the model correctly identified points in every class.

		ACTUAL VALUES	
		Negative	Positive
PREDICTED VALUES	Negative	True Negative (TN)	False Positive (FP)
	Positive	False Negative (TN)	True Positive (TP)

Figure 4.4: Confusion Matrix.

**4.5.2. Evaluation metrics.** There are four main metrics used to evaluate a classification models, which are: accuracy, precision, recall, and F-score [81]. These metrics are used to evaluate the goodness of measure for each of our models. They are:

- **Accuracy:** the accuracy is the proportion of correct predictions (both true positives and true negatives). Accuracy has a range from 0 – 1, with 0 being the worst accuracy and 1 being the best accuracy. Accuracy can be calculated using the following equation [82]:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

- **Precision:** precision is the number of correct results divided by the number of all returned results. Precision has a range from 0 – 1, with 0 being the worst precision and 1 being the best precision. Precision can be calculated using the following equation [83]:

$$Precision = \frac{TP}{TP + FP} \quad (2)$$

- **Recall:** Recall is the true positive rate or *sensitivity*, which measures the proportion of positives that are correctly identified. Recall has a range from 0 – 1, with 0 being the worst recall and 1 being the best recall. Recall can be calculated using the following equation [83]:

$$Recall = \frac{TP}{TP + FN} \quad (3)$$

- **F-score:** F-score is a measure that combines precision and recall is the harmonic mean of precision and recall. F-score has a range from 0 – 1, with 0 being the worst F-score and 1 being the best F-score. F-score can be calculated using the following equation [84]:

$$F - score = \frac{Precision \cdot Recall}{Precision + Recall} \quad (4)$$

In our experiments, since it is a multi-class classification problem, the above metrics were calculated for each of the class (18 devices), and an average of all of them was then taken.

- **ROC Curves:** An ROC curve (Receiver Operating Characteristic) is a graph showing the performance of a classification model [85]. This curve plots two parameters: True Positive Rate versus False Positive Rate. When the Area Under Curve (AUC) [86] of an ROC curve is close to 1, this indicates an excellent model. An example of an ROC curve with  $AUC = 1$  is shown in Figure 4.5. Typically, ROC curves are used in binary classification problems (when there are two classes, and the model is to predict one of the two). In order to leverage ROC curves for multi-class classification problem, the comparison is converted to binary. That is, it will be one class versus all other class. In this way we will be left with a graph with 18 plots, one for each class of devices.

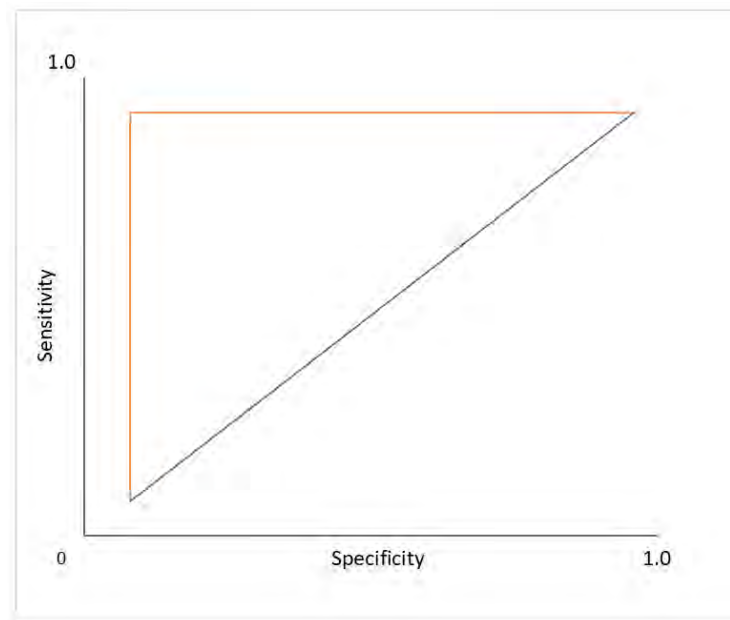


Figure 4.5 ROC Curve.

- **K-fold validation:** K-fold is model validation technique which tests the model's ability to predict new data that was not used in estimating it, in order to flag problems like overfitting [87]. It will split dataset into k consecutive folds. Each fold is then used a test set once while the k - 1 remaining folds form the training set [88]. Figure 4.6 shows an example of K-Fold where number of folds = 5.



Figure 4.6: K-Fold with 5 Folds Example.

- Micro and Macro Average ROC (TP and FP):** When plotting the ROC curves, there are two values, the micro and macro average of the True Positive and False Positive rates. By definition, micro-average of the True Positive Rate is the sum of the of the True Positives by the total number of points [89]. It aims to aggregate all True Positives and show a representation of the correctly identified points with respect to the total number of points in all classes. Macro-average on the other hand, will calculate the ratio of the True Positives for each class individually and then taking the average of all of them [90]. Micro-average will provide equal importance to each sample, whereas macro-average provides equal importance to each class.

The overall summary of the methodology is as follows:

1. Data is gathered from 18 ESP32 device
2. Data has been processed and the Interarrival time feature has been extracted
3. Data has been converted either to Mel Log coefficients or statistical features
4. Models are built and trained
5. Evaluation criteria is checked against the results of the models to check for goodness of results and overfitting of models

Since in this thesis we are running three main experiments, the traditional machine learning models, the CNN using Mel Spectrogram, and CNN using Binary input, the preparation of this data was performed prior to the experiment.

## Chapter 5. Experimental Design: Setup and Data Gathering

In this chapter, the experimental setup that was done to generate the data from the devices is described. In section 5.1, the experimental lab setup and data generation process is described.

### 5.1. Experimental Setup

This experiment is configured in the following manner. The server that runs the broker mainly communicates with the edge device (client) in four ways: turn on, turn off, kill, and check the status of the device. That is, the server can turn on, turn off, kill (disconnect), and check status of the device. It also confirms (acknowledges) the arrival of its commands to the device. To achieve this, there are 4 topics in the MQTT Broker defined:

- */status: checks status of the device (heartbeat)*
- */publishdata: send Turn on and Turn off command*
- */ack: acknowledge the message is received*

This is implemented on the server side by assigning 2 variables and applying the logic that follows it:

- **onTimings:** contains Turn ON Times [9 AM & 12 AM]
- **offTimings:** contains Turn OFF Times [6 PM & 5 PM]

The status of the device is sent to the server every 1 second (heartbeat). This is performed using a separate topic */status*. When a command is published, an acknowledgement is sent back to confirm arrival of the command. This is performed using a separate topic */ack*. This data has been collected for 18 ESP32 devices over a period of approximately 28 days.

To generate the data, ESP run a program that involves receiving commands from the server. These four **commands** are: **Status, Kill, Turn on, and Turn off**.

These commands are described in Table 5.1:

Table 5.1: MQTT Commands.

Command	Description
<b>Status</b>	Server asks the ESP (edge device) its current status
<b>Kill</b>	Server shuts down the ESP (edge device)
<b>Turn ON</b>	Server turns on the ESP (edge device)
<b>Turn OFF</b>	Server turns off the ESP (edge device)

Since MQTT leverages a publish-subscribe communication model, the server will run the **broker** and the edge devices will publish & subscribe through it. To achieve this, **three topics** are created and are: **/status**, **/kill**, **/publishdata**. These topics are described in Table 5.2.

Table 5.2: MQTT Topics.

Command	Description
<b>/status</b>	Heartbeat
<b>/kill</b>	Topic for server shutting down of ESP (edge device)
<b>/publishdata</b>	Topic for server to publish its Turn ON & Turn OFF commands (edge device)

The lab setup is shown in Figure 5.1:

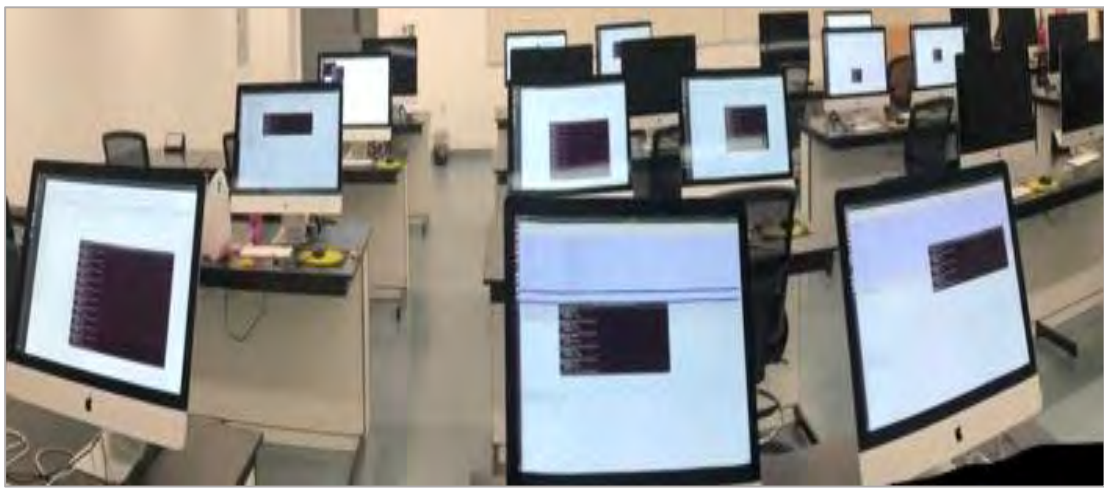


Figure 5.1: Lab Setup.

Experimental Setup view is shown in Figure 5.2:

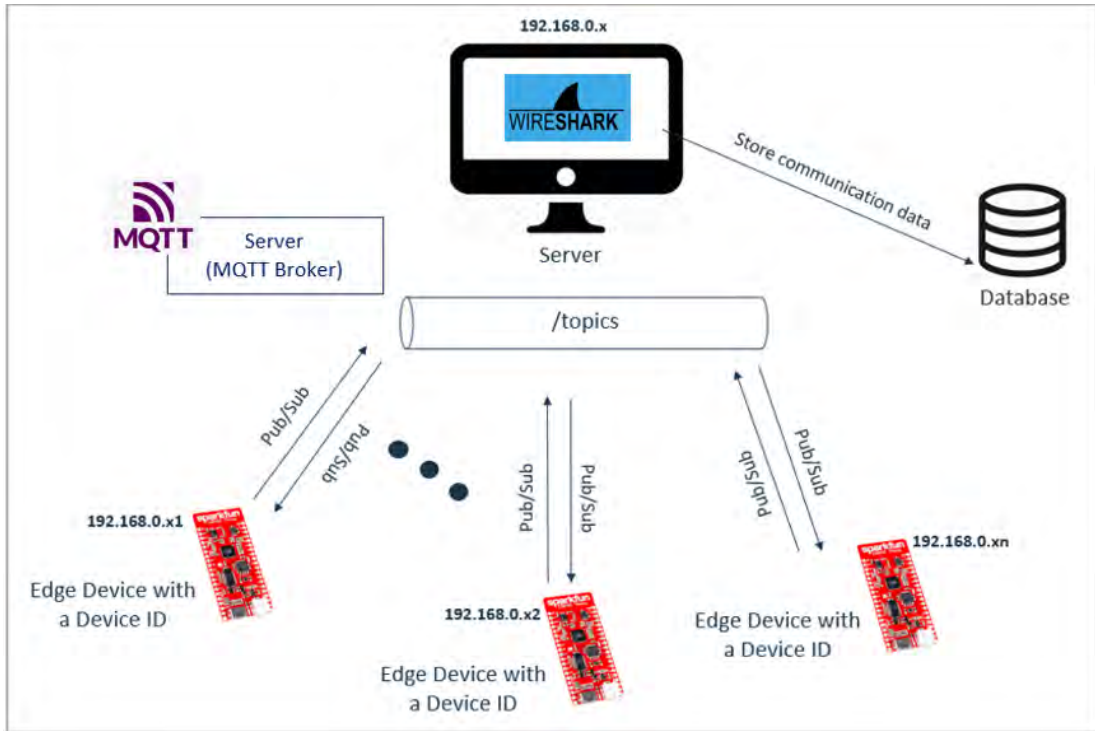


Figure 5.2: Experimental Setup.

The feature that will be experimented to uniquely identify & distinguish is the **Inter-Aarrival Time (IAT)** between the network packets. Wireshark has been leveraged to capture the packets for the setup shown above [91].

## Chapter 6. Traditional Machine Learning Models: Experiments and Results

In this chapter, the traditional machine learning models that were fit are presented. In the first section of this chapter, we discuss the data processing phase, where data was converted before using it to fit the model. In the following section, we present in detail each of the models built. For each model we discuss the results and goodness of measures. The models built were: Random Forest, Bayesian, Support Vector Machine (SVM), Light Gradient Boosting Machine (LightGBM), Gradient Boost. and XG-Boost. All models were built using *scikit-learn*.

### 6.1. Data Processing

The statistical features of the IAT captured from the 18 devices were used as an input to the machine learning models. The total number of points generated from the data was around 14,000,000. Eight statistical features from the IAT were captured, which are:

- Mean
- Maximum
- Minimum
- Variance
- Kurtosis
- Skewness
- Median
- Standard Deviation

The above statistical features were calculated for every second. Therefore, every second is now represented as a row of the 8 values above. This was performed for all the data for all the devices. This was used as an input to the models. In our experiment, the data is labelled as we know which data point belongs to which device. Therefore, every row was considered a second and is labelled with the device number (0 – 17). The data was then normalized between (0 – 1) to keep the data values in same range. A snapshot for a few rows of data with their labels is shown in Figure 6.1.

	Mean	Max	Min	Var	Kurtosis	Skewness	Median
26720	0.319218	0.059579	0.0	0.027396	0.005019	0.038071	0.0
35807	0.332858	0.229562	0.0	0.145189	0.073203	0.265051	0.0
8791	0.461584	0.063499	0.0	0.033448	0.004458	0.051902	0.0
56231	0.296990	0.062084	0.0	0.025075	0.003465	0.041045	0.0
36591	0.339456	0.046892	0.0	0.028128	0.004527	0.051246	0.0
...	...	...	...	...	...	...	...
53724	0.358295	0.062831	0.0	0.028800	0.002481	0.042078	0.0
19099	0.336861	0.509333	0.0	0.308891	0.154719	0.387078	0.0
43167	0.366564	0.059661	0.0	0.028820	0.004422	0.049837	0.0
76164	0.439057	0.040290	0.0	0.036679	0.004609	0.061127	0.0
6571	0.405885	0.063335	0.0	0.031908	0.004453	0.051743	0.0
	Std	Device					
26720	0.205140	5					
35807	0.381035	7					
8791	0.605905	1					
56231	0.336995	11					
36591	0.325767	7					
...	...	...					
53724	0.375291	10					
19099	0.555777	3					
43167	0.355063	8					
76164	0.571342	15					
6571	0.603135	1					

Figure 6.1: Sample Statistical Data for Traditional Machine Learning Models.

The data was then split into training and testing sets, dividing the data into 70% training and 30% testing. Once the training and testing sets were ready, the models were trained and used to predict the devices. The input is consisting of data of around 7 days.

## 6.2. Machine Learning Models

In this section, each of the machine learning models and their results are presented. For each of the algorithms, the following is run:

- Using the **statistical input** described in section 6.1
- Hypertuning the model with statistical input
- Using the log of **Mel Spectrum** input (the input to CNN) as described in Chapter 7
- Hypertuning the model with Mel input

The results of each are presented in terms of the 10 fold accuracies, precision, recall, and F1-score, along with the respective ROC curves and confusion matrix. A summary of all the runs is presented towards the end of this section.

**6.2.1. Random forest model.** Random Forest is one of the classification algorithms that leverages decision trees to perform a classification. Random Forest builds multiple decision trees and merges them together to get a more accurate and stable prediction. A tree is built for each of the samples from the dataset and receive a prediction result from every tree. Voting is performed for every selected result and at last, the most voted prediction is used as a final prediction of the class [92].

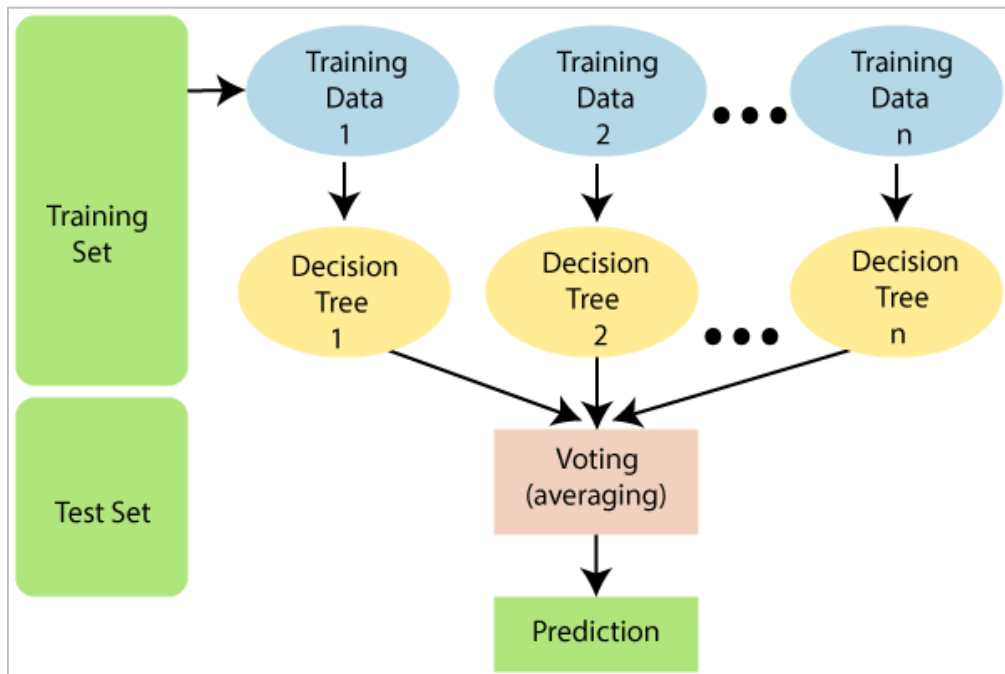


Figure 6.2 Random Forest Classifier [93].

**6.2.1.1. Random forest with statistical input.** In our experiment, Random Forest was built using 10 trees. The model was first tested with fewer number of trees, 4-10, and 10 provided the highest accuracy, therefore it was selected. The input to the model is the statistical values of the data. The accuracy of the model was performed using goodness of measures metrics and K-Fold. In Table 6.1, the experiment parameters are shown. The default hyperparameters were used to train the model.

**Experiment parameters:**

Table 6.1: Random Forest Experiment Parameters.

Input parameters	Values
Random Forest number of trees	10
Training and Testing splits	Training: 70% Testing: 30%

The goodness of measures classification report of the results showing the recall, precision, and F1-Score values, the Confusion Matrix, the ROC Curve, and the K-fold results for the Random Forest Model are presented.

**Goodness of Measure Classification Report:**

Table 6.2 presents the precision, recall, and F1-score values for each of the devices. The micro, macro, and weighted average of all the classes is shown too. A macro-average is the average of all the values for each individual class. A micro-average will aggregate the contributions of all class. That is, it will compute the positive contributions (True Positives) for all classes divided by the total number of points.

Table 6.2: Random Forest Classification Results (Statistical Input).

Device	Precision	Recall	F1-Score
0	0.96	0.96	0.96
1	0.91	0.91	0.91
2	0.98	0.99	0.99
3	0.89	0.91	0.9
4	0.95	0.91	0.93
5	1	1	1
6	0.95	0.94	0.95
7	0.94	0.95	0.95
8	1	1	1
9	0.96	0.92	0.94
10	0.95	0.95	0.95
11	0.94	0.96	0.95
12	0.99	0.96	0.97
13	0.94	0.93	0.93
14	0.99	0.99	0.99
15	0.95	0.98	0.97
16	0.99	0.99	0.99
17	0.93	0.97	0.95
<b>Micro Average</b>	0.96	0.96	0.96
<b>Macro Average</b>	0.96	0.96	0.96

Since the values of the metrics shown is 0.96, this is an indication that the model is a good model and is able to predict the devices with high accuracy. However, in order to verify that is model has not been overfitted, K-fold is performed.

### Confusion Matrix:

The confusion matrix in Figure 6.3 shows the correctly and incorrectly classified devices that the Random Forest classifier predicted.

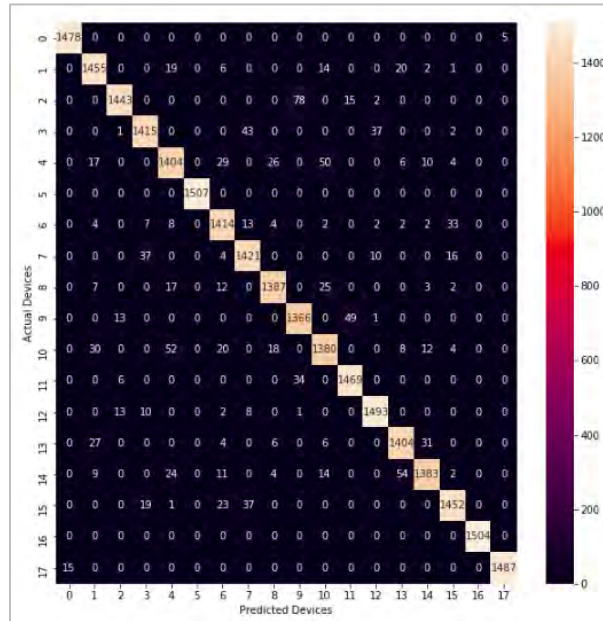


Figure 6.3: Confusion Matrix for Random Forest (Statistical Input).

### ROC Curve:

Figure 6.4 shows the ROC plots for the Random Forest Classifier. It presents the plot for each device against all. From the curve, one can interpret that for all of the devices the True Positive rate is much higher than the False Positive rate.

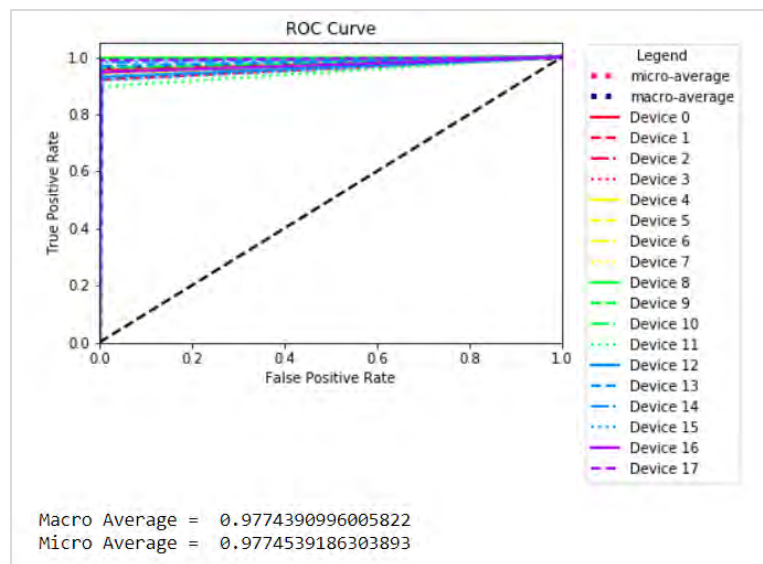


Figure 6.4: ROC Curve for Random Forest Classifier (Statistical Input).

In Table 6.3, the AUC values for each of the curves are shown. All of the AUC values are above 0.95 which is an indication of a good model, as more than 95% percent of the devices were correctly identified.

Table 6.3: Random Forest Classifier (Statistical Input) AUC Values.

Device Number	AUC value
Device 0	0.9749
Device 1	0.9593
Device 2	0.9717
Device 3	0.9960
Device 4	0.9720
Device 5	0.9993
Device 6	0.9701
Device 7	0.9715
Device 8	0.9986
Device 9	0.9949
Device 10	0.9801
Device 11	0.9463
Device 12	0.9638
Device 13	0.9828
Device 14	0.9898
Device 15	0.9559
Device 16	0.9724
Device 17	0.9943
<b>Macro Average</b>	<b>0.9774</b>
<b>Micro Average</b>	<b>0.9774</b>

### **K-Fold:**

In order to validate that the model has not been over-fitted, K-Fold is implemented. The results of the mean and standard deviation for the 10 fold runs are shown in Table 6.4. It can be shown that all folds resulted in an accuracy, precision, recall, and F1-Score values are 0.95 on average, and are close to the values generated when running the model, which indicates that the model is not over-fitted.

Table 6.4: K-Fold Results for Random Forest (Statistical Input) - Mean (Standard Deviation).

Metric	Accuracy	Precision	Recall	F1-score
<b>Mean</b>	0.9501 (0.0036)	0.9505(0.0036)	0.9501(0.0036)	0.9501(0.0036)

### Hypertuning:

In order to achieve better results, hypertuning for the Random Forest model was performed. The tuning parameters and the best result are shown in Table 6.5.

Table 6.5: Random Forest (Statistical Input) Hypertuning.

Metric	Options	Best Result
<b>n_estimators</b>	min 10 max 100 step 10	25
<b>max_features</b>	['auto', 'sqrt']	auto
<b>max_depth</b>	10, 110	24
<b>min_samples_split</b>	[2, 5, 10]	2
<b>min_samples_leaf</b>	[1, 2, 4]	2
<b>bootstrap</b>	[True, False]	False

The model was trained, and 10 fold validation was performed. The results of the mean of the 10 fold validation for each of the metrics is shown in Table 6.6. The hypertuning improved the results and increased the F1 -score from 0.95 to 0.96.

Table 6.6: K-Fold for (Statistical Input) Random Forest - Mean (Standard Deviation) After Hypertuning.

Metric	Accuracy	Precision	Recall	F1-score
<b>Mean</b>	0.9621 (0.0044)	0.9637 (0.0045)	0.9641 (0.0040)	0.9673 (0.0038)

**6.2.1.2. Random forest with mel input.** The Random Forest model was then used to train a Random Forest Classifier, however, using the same input as the Mel input (same as CNN input).

### Goodness of Measure Classification Report:

Table 6.7 presents the precision, recall, and F1-score values for each of the devices of the model run. It can be seen that the average results for these metrics are around 0.91.

Table 6.7: Random Forest (MelInput) Classification Results.

Device	Precision	Recall	F1-Score
0	1	1	1
1	0.93	0.92	0.93
2	0.9	0.91	0.9
3	0.97	0.96	0.96
4	0.87	0.88	0.88
5	0.87	0.89	0.88
6	0.9	0.89	0.9
7	0.92	0.9	0.91
8	0.95	0.96	0.95
9	0.9	0.88	0.89
10	0.89	0.88	0.89
11	0.83	0.83	0.83
12	0.96	0.97	0.96
13	1	1	1
14	0.9	0.87	0.88
15	0.81	0.82	0.82
16	0.91	0.93	0.92
17	0.89	0.92	0.91
<b>Micro Average</b>	0.91	0.91	0.91
<b>Macro Average</b>	0.91	0.91	0.91

### **Confusion Matrix:**

The confusion matrix in Figure 6.5 shows the correctly and incorrectly classified devices that the Random Forest classifier predicted. From the results, it can be shown that there are huge number of devices that are incorrectly identified, indicating that Random Forest model does not work well when the input provided is the Mel input (same as CNN input).

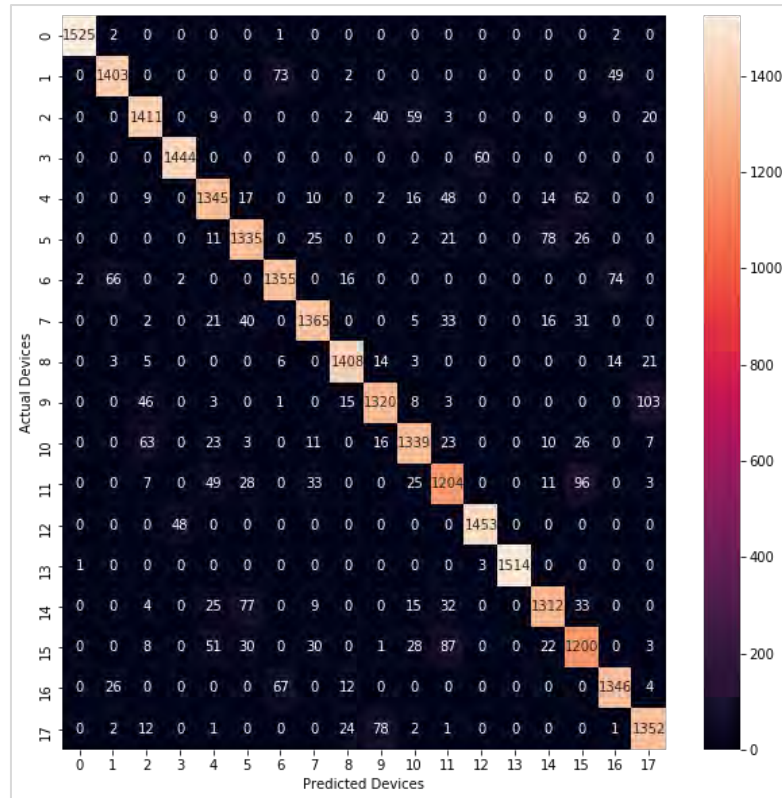


Figure 6.5: Confusion Matrix (MelInput) for Random Forest.

**ROC Curve:**

Figure 6.6 shows the ROC plots for the Random Forest Classifier. It presents the plot for each device against all.

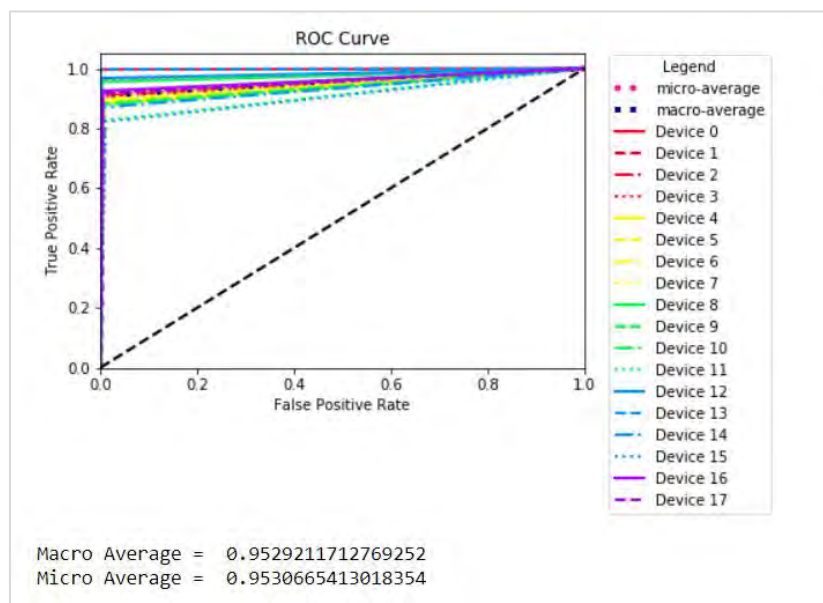


Figure 6.6: ROC Curve for Random Forest (MelInput) Classifier.

### K-Fold:

In order to validate that the model has not been over-fitted, K-Fold is implemented. The results of the mean and standard deviation for the 10 fold runs are shown in Table 6.8.

Table 6.8: K-Fold for Random Forest (Mel Input) - Mean (Standard Deviation).

Metric	Accuracy	Precision	Recall	F1-score
Mean	0.9042 (0.0050)	0.9047 (0.0048)	0.9043 (0.0050)	0.9044 (0.0049)

### Hypertuning:

In order to achieve better results, hypertuning for the Random Forest model was performed. The tuning parameters and the best result are shown in Table 6.11.

Table 6.9: Random Forest (Mel Input) Hypertuning.

Metric	Options	Best Result
n_estimators	min 10 max 100 step 10	20
max_features	['auto', 'sqrt']	auto
max_depth	[10, 110] step = 10	40
min_samples_split	[2, 5, 10]	2
min_samples_leaf	[1, 2, 4]	2
bootstrap	[True, False]	False

The model was trained, and 10 fold validation was performed. The results of the mean of the 10 fold validation for each of the metrics is shown in Table 6.10. The hypertuning improved the results and increased the F1 -score from 0.90 to 0.93.

Table 6.10: K-Fold for Random Forest (Mel Input) - Mean (Standard Deviation) After Hypertuning.

Metric	Accuracy	Precision	Recall	F1-score
Mean	0.9232 (0.0050)	0.9235 (0.0050)	0.9363 (0.0051)	0.9356 (0.0051)

**Conclusion:** From the results, it can be interpreted that using the statistical input, the tuned model had a better result (F1 -score **0.95**) in comparison with the tuned model with Mel input (F1 -score **0.93**).

**6.2.2. Bayesian model.** Bayesian classifiers build the model based on statistical models that are built upon the Bayes Theorem. Bayes Theorem states that probability of an event occurring based on historical or prior knowledge. For example, if it is known that the risk of worsening health conditions increases with the age of the person, then the model will assess the risk of the persons health with regards to their age. In machine learning, the Bayesian model will perform the Maximum Likelihood Estimation iteratively to update the model parameters to increase the probability of seeing input data when already seen model the parameters [94]. For the Bayesian model, hypertuning was not performed as both models with both inputs resulted in very weak results and no work on them will be taken forward.

**6.2.2.1. Bayesian model with statistical input.** In this section, the model with statistical input and the results are shown. In Table 6.11, the experiment parameters are shown. The default hyperparameters were used to train the model.

**Experiment parameters**

Table 6.11: Bayesian Classifier (Statistical Input) Experiment Parameters.

Input parameters	Values
Method	Gaussian
Training and Testing splits	Training: 70% Testing: 30%

The goodness of measures classification report of the results showing the recall, precision, and F1-Score values, the Confusion Matrix, the ROC Curve, and the K-fold results for the Bayesian Model are presented.

**Goodness of Measure Classification Report:**

Table 6.12 presents the precision, recall, and F1-score values for each of the devices. The micro, macro, and weighted average of all the classes is shown too. A macro-average is the average of all the values for each individual class. A micro-average will aggregate the contributions of all class. That is, it will compute the positive contributions (True Positives) for all classes divided by the total number of points.

Table 6.12: Bayesian Classifier (Statistical Input) Classification Results.

Device	Precision	Recall	F1-Score
0	0.41	0.61	0.49

1	0.41	0.71	0.52
2	0.05	0.01	0.02
3	0.05	0	0.01
4	0	0	0
5	0	0	0
6	0.26	0.11	0.15
7	0	0	0
8	0.11	0.07	0.08
9	0.18	0.61	0.28
10	0.06	0.08	0.06
11	0.03	0.02	0.02
12	0	0	0
13	0.22	0.23	0.22
14	0.2	0.8	0.32
15	0.13	0.2	0.16
16	0.96	0.5	0.66
17	0.04	0	0
<b>Micro Average</b>	0.22	0.22	0.22
<b>Macro Average</b>	0.17	0.22	0.17

Since the values of the metrics shown is very low (the micro average is around 0.22), this is an indication that the model is a poor model and is unable to predict the devices with high accuracy. However, as an extra check, K-fold is performed.

#### **Confusion Matrix:**

The confusion matrix in Figure 6.7 shows the correctly and incorrectly classified devices that the Bayesian classifier predicted. From the results, it can be shown that there are huge number of devices that are incorrectly identified, indicating that Random Forest model does not work well when the input provided is the Mel input (same as CNN input).

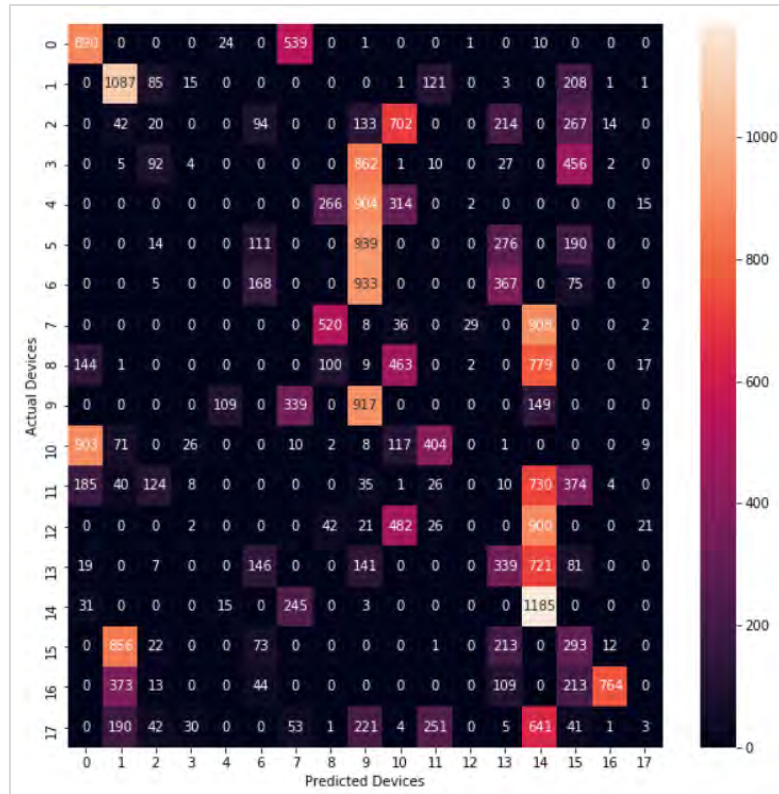


Figure 6.7: Confusion Matrix for Bayesian (Statistical Input).

**ROC Curve:**

Figure 6.8 shows the ROC plots for the Bayesian Classifier. It presents the plot for each device against all. From the curve, one can interpret that for Devices 1, 14, and 16 had the best AUC values (True Positive ratio to False Positive) among the other devices, where for other devices the model performed badly.

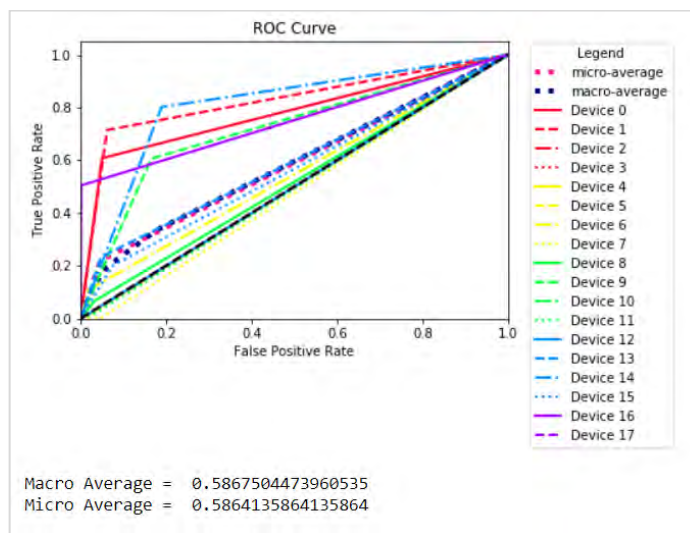


Figure 6.8: ROC Curve for Bayesian Classifier (Statistical Input).

In Table 6.13, the AUC values for each of the curves are shown. It can be shown that the highest AUC values (around 0.8) were corresponding for devices 1, 14, and 16. Other devices AUC values range between 0.4, 0.5, and 0.7 values, indicating that for some devices, more than half of the points for these devices were falsely classified.

Table 6.13: Bayesian Classifier (Statistical Input) AUC Values.

Device Number	AUC value
Device 0	0.7787
Device 1	0.8262
Device 2	0.4988
Device 3	0.4998
Device 4	0.4971
Device 5	0.5000
Device 6	0.5451
Device 7	0.4768
Device 8	0.5167
Device 9	0.7202
Device 10	0.4984
Device 11	0.4925
Device 12	0.4993
Device 13	0.5926
Device 14	0.8059
Device 15	0.5624
Device 16	0.7513
Device 17	0.4997
Macro Average	0.5867
Micro Average	0.5864

### K-Fold:

In order to validate that the model has not been over-fitted, K-Fold is implemented. The results of the 10-fold runs are shown in Table 6.14. It can be shown that all folds resulted in an accuracy, precision, recall, and F1 -Score values are all very low, 0.2 on average. This means that after K-fold the values match the model results, and there was no over-fitting. The conclusion is that Bayesian did not perform well at predicting the devices.

Table 6.14: K-Fold for Bayesian (Statistical Input) - Mean (Standard Deviation).

Metric	Accuracy	Precision	Recall	F1-score
<b>Mean</b>	0.2196 (0.0024)	0.1727 (0.0035)	0.2194 (0.0023)	0.1683 (0.0016)

**6.2.2.2. Bayesian model with mel input.** The model was also trained using the Mel input (CNN model). The goodness of measures classification report of the results showing the recall, precision, and F1-Score values, the Confusion Matrix, the ROC Curve, and the K-fold results for the Bayesian Model are presented.

**Goodness of Measure Classification Report:**

Table 6.15 presents the precision, recall, and F1-score values for each of the devices.

Table 6.15: Bayesian Classifier (Mel Input) Classification Results.

Device	Precision	Recall	F1-Score
0	0.49	0.54	0.51
1	0.39	0.7	0.5
2	0.07	0.02	0.03
3	0.05	0	0.01
4	0	0	0
5	0	0	0
6	0.27	0.11	0.16
7	0	0	0
8	0.11	0.07	0.09
9	0.22	0.6	0.32
10	0.07	0.11	0.09
11	0.04	0.02	0.03
12	0	0	0
13	0.24	0.23	0.23
14	0.17	0.91	0.28
15	0.12	0.21	0.15
16	0.98	0.35	0.51
17	0	0	0
<b>Micro Average</b>	0.21	0.21	0.21
<b>Macro Average</b>	0.18	0.21	0.16

Since the values of the metrics shown is very low, this is an indication that the model is a poor model and is unable to predict the devices with high accuracy. However, as an extra check, K-fold is performed (shown).

### Confusion Matrix:

The confusion matrix in Figure 6.9 shows the correctly and incorrectly classified devices that the Bayesian classifier predicted. From the results, it can be shown that there are huge number of devices that are incorrectly identified, indicating that Bayesian model does not work well when the input provided is the Statistical input data.

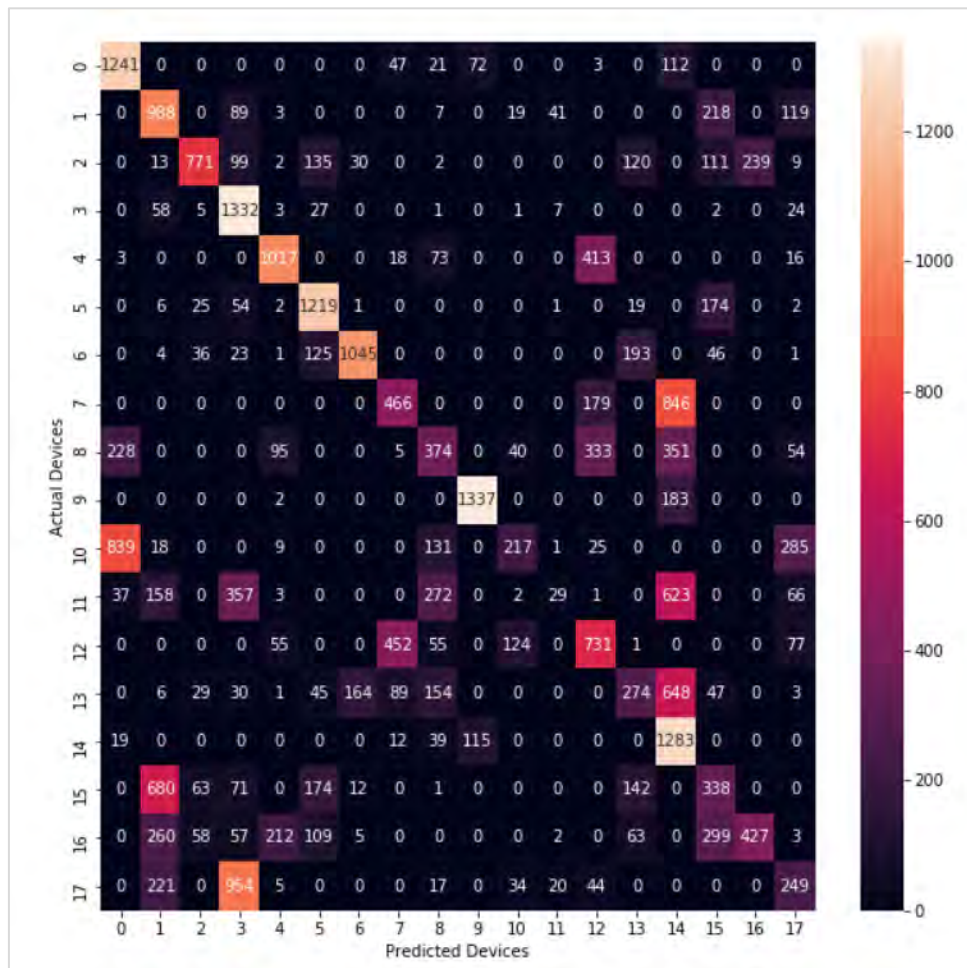


Figure 6.9: Confusion Matrix (Mel Input) for Bayesian Classifier.

### ROC Curve:

Figure 6.10 shows the ROC plots for the Bayesian Classifier. It presents the plot for each device against all showing a high False Positive rate.

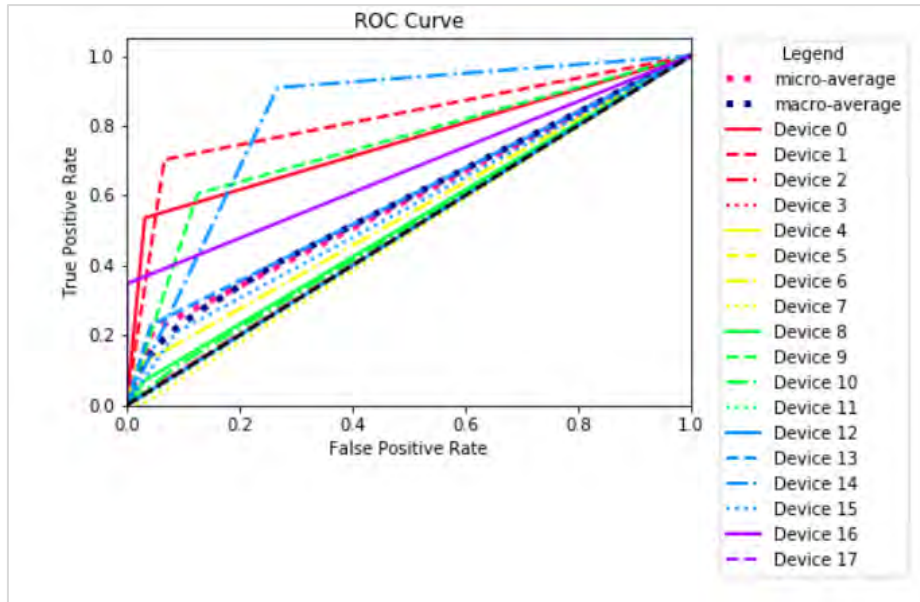


Figure 6.10: ROC Curve for Bayesian Classifier (Mel Input).

### K-Fold:

In order to validate that the model has not been over-fitted, K-Fold is implemented. The results of the 10-fold are shown in Table 6.16.

Table 6.16 K-Fold for Bayesian (Mel Input) - Mean (Standard Deviation).

Metric	Accuracy	Precision	Recall	F1-score
<b>Mean</b>	0.1794 (0.0023)	0.2169 (0.0023)	0.1667 (0.0023)	0.2169 (0.0024)

**Conclusion:** From the results, it can be interpreted that using the Mel input, the tuned model had a better result (F1-score **0.21**) in comparison with the model with the statistical input (F1-score **0.16**).

**6.2.3. Support vector machine model.** Support Vector Machines (SVM) are one of the supervised learning models used for classification and regression problems. SVM models takes the input data points and will output the respective hyperplane that will separate the classes. Often this hyperplane is called the decision boundary and acts as a line that separates on either side of the plane. Figure 6.11 [95] shows an SVM example. SVM algorithms use mathematical functions, that are known as *kernels*. These functions are used to take in input points and transform it into a required form. In this thesis, we perform SVM using two kernels: Linear and Radial Basis Function (RBF). Linear kernels are leveraged when the data is linearly separable and can be separated using a line. RBF kernels are leveraged for non-linear separations [95].

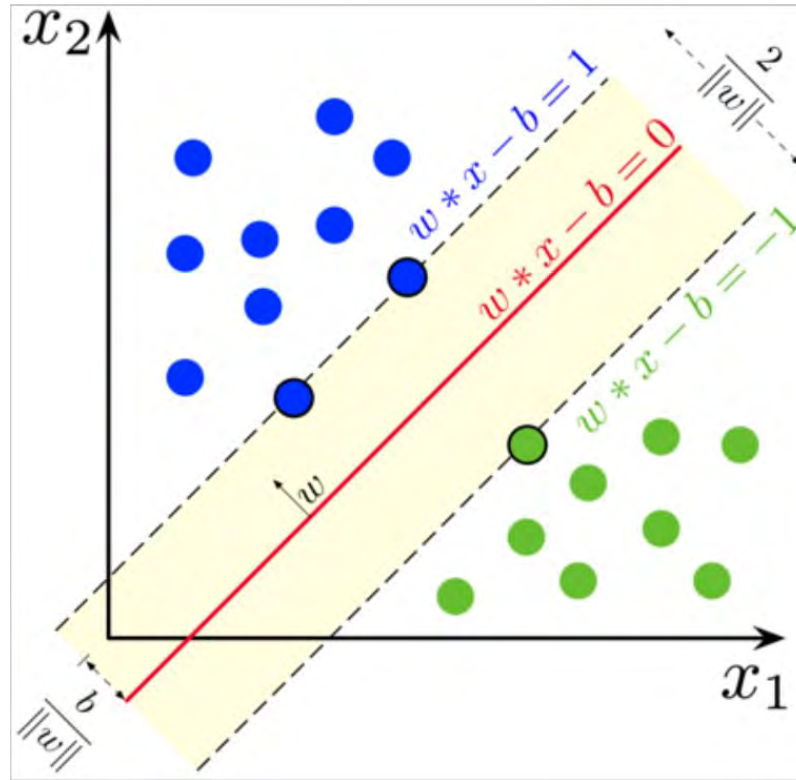


Figure 6.11: SVM Example [95].

**6.2.3.1. Support vector machine - linear with statistical input.** In this section, the model trained with statistical input, and the goodness of measures classification report of the results showing the recall, precision, and F1-Score values, the Confusion Matrix, the ROC Curve, and the K-fold results for the Support Vector Machine (linear) model are presented. The experimental parameters are shown in Table 6.17.

**Experiment parameters:**

Table 6.17: SVM Linear Classifier Experiment Parameters (Statistical Input)

Input parameters	Values
kernel	Linear
Training and Testing splits	Training: 70% Testing: 30%

**Goodness of Measure Classification Report:**

Table 6.18 presents the precision, recall, and F1-score values for each of the devices. The results show a micro average score of 0.56 for all of the metrics (precision, recall, and F1-score).

Table 6.18: SVM Linear Classifier Classification Results (Statistical Input).

Device	Precision	Recall	F1-Score
0	0.52	0.88	0.65
1	0.43	0.57	0.49
2	0.73	0.53	0.61
3	0.56	0.56	0.56
4	0.69	0.74	0.71
5	0.7	0.81	0.75
6	0.84	0.72	0.77
7	0.55	0.36	0.44
8	0.37	0.31	0.34
9	0.9	0.89	0.9
10	0.57	0.24	0.33
11	0.57	0.5	0.53
12	0.57	0.68	0.62
13	0.35	0.18	0.24
14	0.42	0.85	0.56
15	0.31	0.28	0.29
16	0.67	0.27	0.39
17	0.53	0.73	0.62
<b>Micro Average</b>	0.56	0.56	0.56
<b>Macro Average</b>	0.57	0.56	0.54

From the results, one can interpret that the model is poor at performing the classification. This is expected since the data cannot be linearly separated.

**Confusion Matrix:**

The confusion matrix in Figure 6.12 shows the correctly and incorrectly classified devices that the SVM classifier predicted. From the results, it can be shown that there are huge number of devices that are incorrectly identified, indicating that SVM Linear model does not work well when the input provided is the Statistical input data.

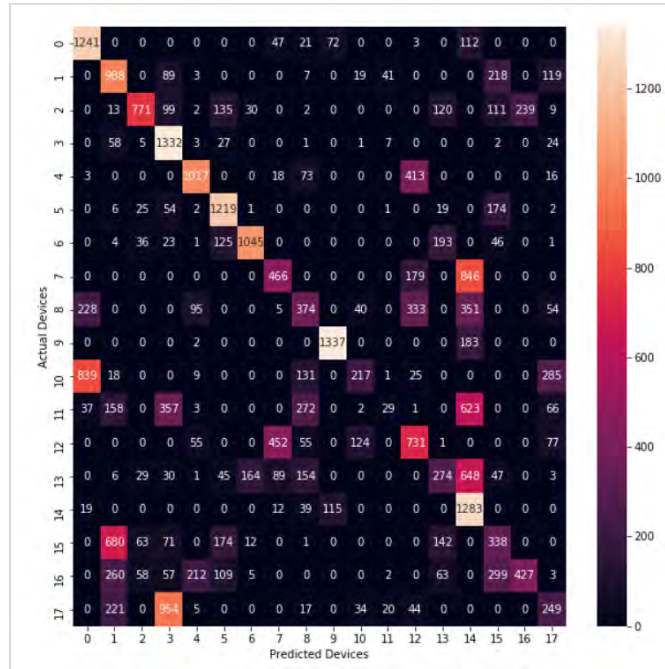


Figure 6.12: Confusion Matrix for SVM – Linear (Statistical Input).

**ROC Curve:**

Figure 6.13 shows the ROC plots for the SVM Linear Classifier. It presents the plot for each device against all. From the curve, one can interpret that for some devices (9, 10, 12, 13, 15, 5, 17) the model performed badly. The result of devices the model has performed better (higher AUC values). A clearer interpretation of this curve is shown in that shows the AUC values.

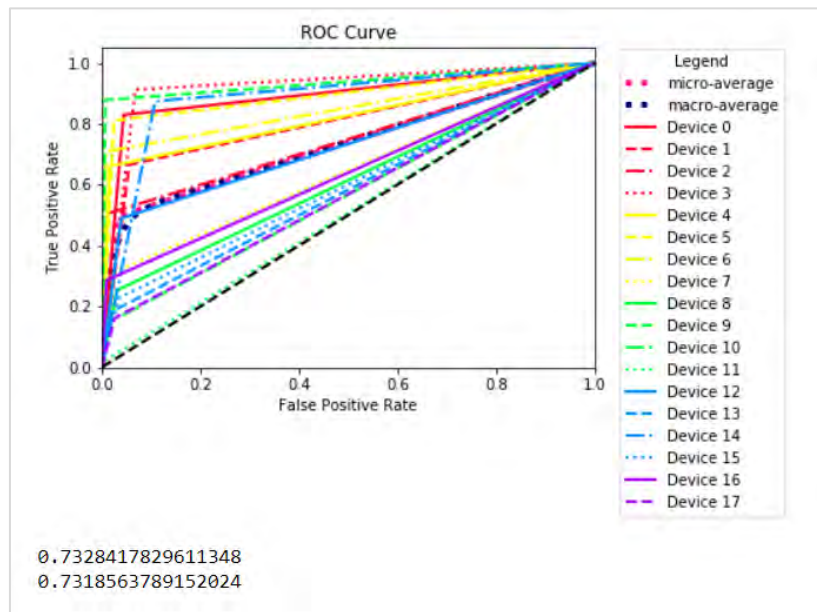


Figure 6.13: ROC Curve for SVM Linear Classifier (Statistical Input).

In Table 6.19, the AUC values for each of the curves are shown. It can be shown that the highest AUC values (around 0.9) were corresponding for devices 3. Other devices AUC values range between 0.5 and 0.7 values, indicating that for some devices, more than half of the points for these devices were falsely classified.

Table 6.19: SVM Linear Classifier AUC Values (Statistical Input).

Device Number	AUC value
Device 0	0.8927
Device 1	0.8050
Device 2	0.7476
Device 3	0.9223
Device 4	0.8225
Device 5	0.8935
Device 6	0.8503
Device 7	0.6441
Device 8	0.6112
Device 9	0.9356
Device 10	0.5668
Device 11	0.5080
Device 12	0.7249
Device 13	0.5814
Device 14	0.8829
Device 15	0.5966
Device 16	0.6381
Device 17	0.5677
Macro Average	0.7328
Micro Average	0.7318

### **K-Fold:**

In order to validate that the model has not been over-fitted, K-Fold is implemented. The results of the 10 fold runs are shown in Table 6.20. It can be shown that all folds resulted in an accuracy, precision, recall, and F1 -Score values are around 0.5 on average. This means that after K-fold the values match the model results, and there was no over-fitting. The conclusion is that SVM Linear model did not perform well at predicting the devices.

Table 6.20: K-Fold for SVM Linear Classifier (Statistical Input) - Mean (Standard Deviation).

Fold No.	Accuracy	Precision	Recall	F1-score
<b>Mean</b>	0.4835 (0.0052)	0.5035 (0.0075)	0.4827 (0.0051)	0.4546 (0.0035)

### Hypertuning:

In order to achieve better results, hypertuning for the SVM Linear model was performed. This ensures that the model is trained with the most optimum parameters. The tuning parameters and the best result are shown in Table 6.21

Table 6.21: SVM Linear Classifier (Statistical Input) Hypertuning.

Metric	Options	Best Result
<b>param_grid</b>	[0.1,1, 10, 100]	1
<b>gamma</b>	[1,0.1,0.01,0.001]	0.01

The model was trained, and 10 fold validation was performed. The results of the mean of the 10 fold validation for each of the metrics is shown in Table 6.22. The hypertuning improved the results and increased the F1-score from 0.45 to 0.53.

Table 6.22: K-Fold for SVM Linear (Statistical Input) - Mean (Standard Deviation) After Hypertuning.

Metric	Accuracy	Precision	Recall	F1-score
<b>Mean</b>	0.5453 (0.0042)	0.5643 (0.0043)	0.5543 (0.0043)	0.5343 (0.0040)

**6.2.3.2. Support vector machine - linear with mel input.** In this section, the model, and the goodness of measures classification report of the results using the Mel input is presented.

### Goodness of Measure Classification Report:

Table 6.23 presents the precision, recall, and F1-score values for each of the devices. The micro average score is around 0.56.

Table 6.23: SVM Linear Classifier (Mel Input) Classification Results.

Device	Precision	Recall	F1-Score
0	0.52	0.88	0.65
1	0.43	0.57	0.49
2	0.73	0.53	0.61
3	0.56	0.56	0.56
4	0.69	0.74	0.71
5	0.7	0.81	0.75
6	0.84	0.72	0.77
7	0.55	0.36	0.44
8	0.37	0.31	0.34
9	0.9	0.89	0.9
10	0.57	0.24	0.33
11	0.57	0.5	0.53
12	0.57	0.68	0.62
13	0.35	0.18	0.24
14	0.42	0.85	0.56
15	0.31	0.28	0.29
16	0.67	0.27	0.39
17	0.53	0.73	0.62
<b>Micro Average</b>	0.56	0.56	0.56
<b>Macro Average</b>	0.57	0.56	0.54

Since the values of the metrics shown is very low, this is an indication that the model is a poor model and is unable to predict the devices with high accuracy. However, as an extra check, K-fold is performed (shown).

**Confusion Matrix:**

The confusion matrix in Figure 6.14 shows the correctly and incorrectly classified devices that the Bayesian classifier predicted. From the results, it can be shown that there are huge number of devices that are incorrectly identified, indicating that SVM Linear model does not work well when the input provided is the Mel input (same as CNN input).

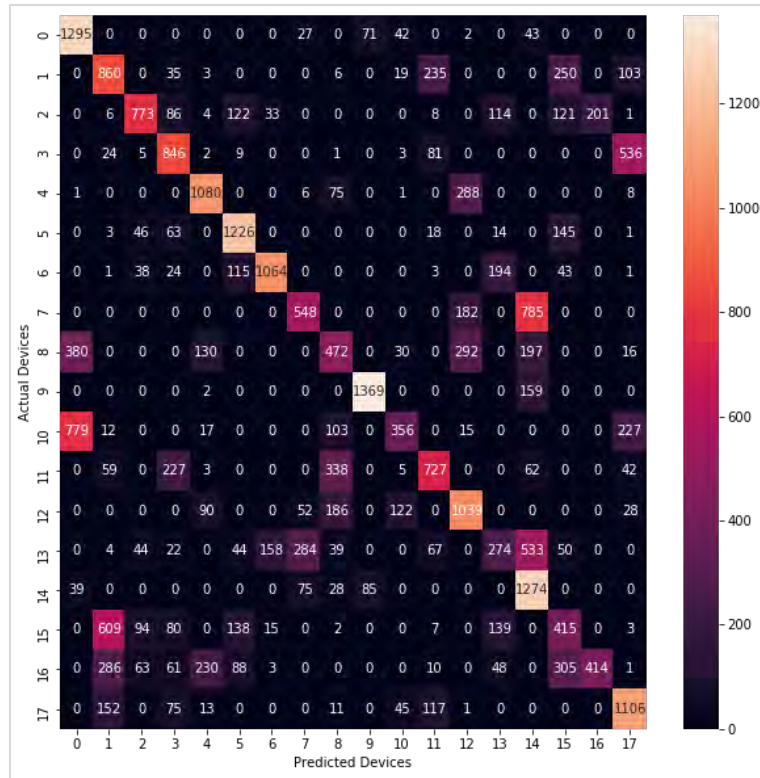


Figure 6.14: Confusion Matrix (MelInput) for SVM Linear.

**ROC Curve:**

Figure 6.15 shows the ROC plots for the Bayesian Classifier. It presents the plot for each device against all.

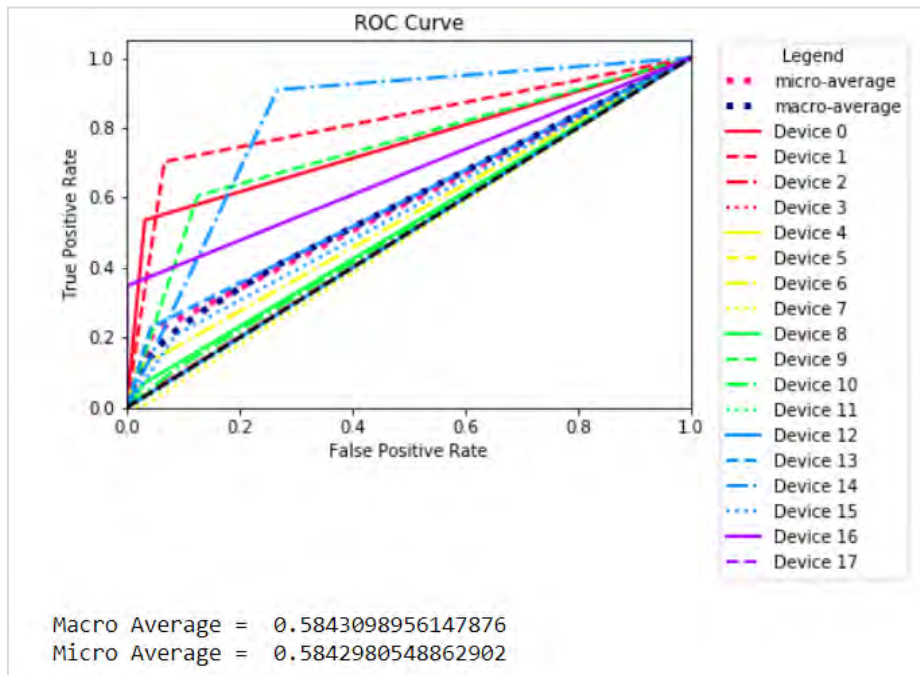


Figure 6.15: ROC Curve for SVM Linear Classifier (MelInput).

### K-Fold:

In order to validate that the model has not been over-fitted, K-Fold is implemented. The results of the 10-fold are shown in Table Table 6.24.

Table 6.24: K-Fold for SVM Linear Classifier (Mel Input)- Mean (Standard Deviation).

Metric	Accuracy	Precision	Recall	F1-score
Mean	0.1794 (0.0023)	0.2169 (0.0023)	0.1667 (0.0023)	0.2169 (0.0024)

### Hypertuning:

In order to achieve better results, hypertuning for the model was performed. The tuning parameters and the best result are shown in Table 6.25.

Table 6.25: SVM Linear Classifier (Mel Input) Hypertuning.

Metric	Options	Best Result
param_grid	[0.1,1, 10, 100]	1
gamma	[1,0.1,0.01,0.001]	1

The model was trained, and 10 fold validation was performed. The results of the mean of the 10 fold validation for each of the metrics is shown in Table 6.26. The hypertuning improved the results and increased the F1 -score from 0.21 to 0.32.

Table 6.26 K-Fold for SVM Linear (Mel Input) - Mean (Standard Deviation) After Hypertuning.

Metric	Accuracy	Precision	Recall	F1-score
Mean	0.2744 (0.0003)	0.3232 (0.0003)	0.2333 (0.0003)	0.3232 (0.0004)

**6.2.3.3. Support vector machine - RBF with statistical input.** In this section, the non-linear model with statistical input, the goodness of measures classification report of the results is shown. In Table 6.27, the experiment parameters are shown. The default hyperparameters were used to train the model.

### Experiment parameters:

Table 6.27: SVM Non-Linear Classifier Experiment Parameters (Statistical Input).

Input parameters	Values
kernel	RBF
Training and Testing splits	Training: 70% Testing: 30%

**Goodness of Measure Classification Report:**

Table 6.28 presents the precision, recall, and F1-score values for each of the devices.

Table 6.28: SVM Non-Linear Classifier Classification Results (Statistical Input).

Device	Precision	Recall	F1-Score
0	0.66	0.91	0.77
1	0.8	0.87	0.83
2	0.84	0.68	0.75
3	0.8	0.87	0.84
4	0.88	0.88	0.88
5	0.73	0.95	0.83
6	0.85	0.82	0.83
7	0.98	0.95	0.97
8	0.81	0.84	0.82
9	0.96	0.95	0.95
10	0.76	0.5	0.6
11	0.86	0.86	0.86
12	0.88	0.89	0.89
13	0.92	0.75	0.82
14	0.9	0.94	0.92
15	0.67	0.78	0.72
16	1	0.72	0.83
17	0.87	0.87	0.87
Micro Average	0.83	0.83	0.83
Macro Average	0.84	0.84	0.83

From the results, one can interpret that the model has performed better than the linear model. The precision, recall, and F1-score values are almost double in comparison to the SVM model with a linear kernel. However, these metrics values are around 0.8 which concludes that this model will not work very well when performing the classification.

### Confusion Matrix:

The confusion matrix in Figure 6.16 shows the correctly and incorrectly classified devices that the SVM classifier predicted. It can be shown that there are more than 3,000 points that were falsely identified.

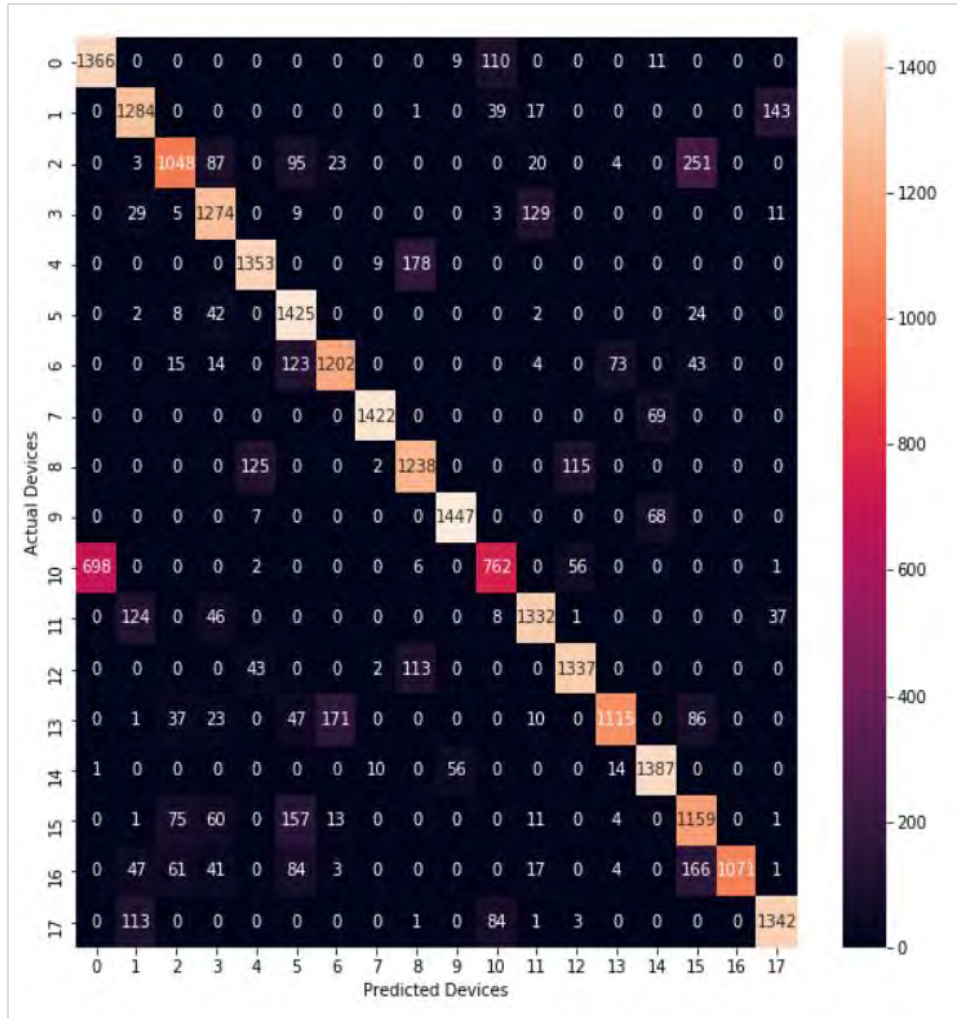


Figure 6.16: Confusion Matrix for SVM - Non-Linear (Statistical Input).

### ROC Curve:

Figure 6.17 shows the ROC plots for the SVM Non-Linear Classifier. It presents the plot for each device against all. From the curve, one can interpret that for some devices (5, 9, 10, 12, 13, 15, 17) the model performed badly. The result of devices the model has performed better (higher AUC values). A clearer interpretation of this curve is shown in that shows the AUC values. In conclusion, it can be shown that there are huge number of devices that are incorrectly identified, indicating that SVM non-linear model does not work well when the input provided is the Statistical input.

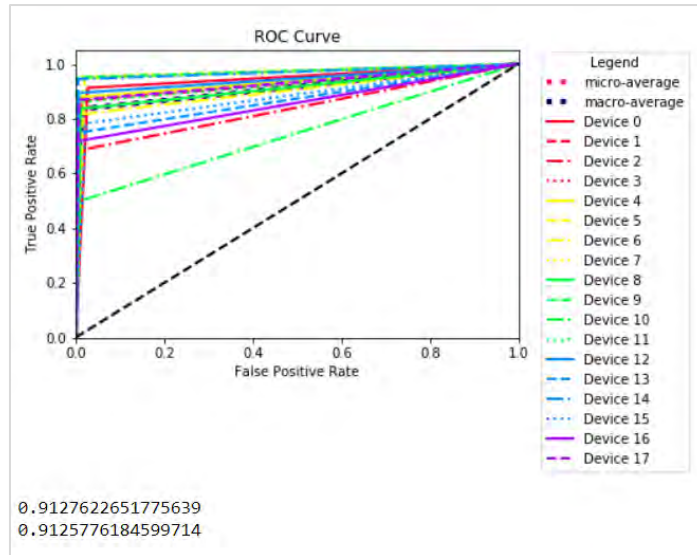


Figure 6.17: ROC Curve for SVM Non-Linear Classifier (Statistical Input).

In Table 6.29, the AUC values for each of the curves are shown. It can be shown that the highest AUC values (around 0.9) were corresponding for most of the devices. The average value is around 0.91 indicating that this model can identify between the devices but with some falsely detection for some.

Table 6.29: SVM Non-Linear Classifier AUC Values (Statistical Input).

Device Number	AUC value
Device 0	0.9429
Device 1	0.9264
Device 2	0.8383
Device 3	0.9302
Device 4	0.9358
Device 5	0.9640
Device 6	0.9036
Device 7	0.9764
Device 8	0.9124
Device 9	0.9741
Device 10	0.7451
Device 11	0.9261
Device 12	0.9437
Device 13	0.8722
Device 14	0.9695
Device 15	0.8801
Device 16	0.8582
Device 17	0.9308

### K-Fold:

In order to validate that the model has not been over-fitted, K-Fold is implemented. The results of the 10 fold runs are shown in Table 6.30.

Table 6.30: K-Fold for SVM Non-Linear Classifier (Statistical Input) - Mean (Standard Deviation)

Metric	Accuracy	Precision	Recall	F1-score
<b>Mean</b>	0.8233 (0.0032)	0.8346 (0.0036)	0.8232 (0.0032)	0.8221 (0.0031)

### Hypertuning:

In order to obtain better results, hypertuning was performed on the model. The tuning parameters and the best result are shown in Table 6.31.

Table 6.31: SVM Non-Linear Classifier (Statistical Input) Hypertuning.

Metric	Options	Best Result
<b>param_grid</b>	[0.1,1,10,100]	0.1
<b>gamma</b>	[1,0.1,0.01,0.001]	0.1

The model was trained, and 10 fold validation was performed. The results of the mean of the 10 fold validation for each of the metrics is shown in Table 6.32. The hypertuning improved the results and increased the F1 -score from 0.82 to 0.84.

Table 6.32: K-Fold for SVM Non-Linear (Statistical Input) - Mean (Standard Deviation) After Hypertuning.

Metric	Accuracy	Precision	Recall	F1-score
<b>Mean</b>	0.8423 (0.0022)	0.8544 (0.0026)	0.8542 (0.0022)	0.8421 (0.0021)

**6.2.3.4. Support vector machine – RBF with mel input.** In this section, the model using the Mel input, the goodness of measures classification report of the results is presented. The experimental parameters are shown in Table 6.33.

### Experiment parameters:

Table 6.33: SVM Non-Linear Classifier Experiment Parameters (with Mel Input).

Input parameters	Values
kernel	RBF
Training and Testing splits	Training: 70% Testing: 30%

**Goodness of Measure Classification Report:**

Table 6.34 presents the precision, recall, and F1-score values for each of the devices.

Table 6.34: SVM Non-Linear Classifier Classification Results (Mel Input).

Device	Precision	Recall	F1-Score
0	0.44	0.54	0.49
1	0.44	0.67	0.53
2	0.74	0.71	0.72
3	0.48	0.26	0.34
4	0.95	0.62	0.75
5	0.52	0.72	0.6
6	0.69	0.56	0.62
7	0.76	0.21	0.33
8	0.27	0.15	0.19
9	1	0.58	0.74
10	0.34	0.12	0.18
11	0.39	0.28	0.32
12	0.12	0.09	0.1
13	0.52	0.15	0.24
14	0.14	0.85	0.24
15	0.35	0.22	0.27
16	0.99	0.41	0.58
17	0.31	0.14	0.2
<b>Micro Average</b>	0.4	0.4	0.4
<b>Macro Average</b>	0.53	0.41	0.41

**Confusion Matrix:**

The confusion matrix in Figure 6.18 shows the correctly and incorrectly classified devices that the SVM classifier predicted.

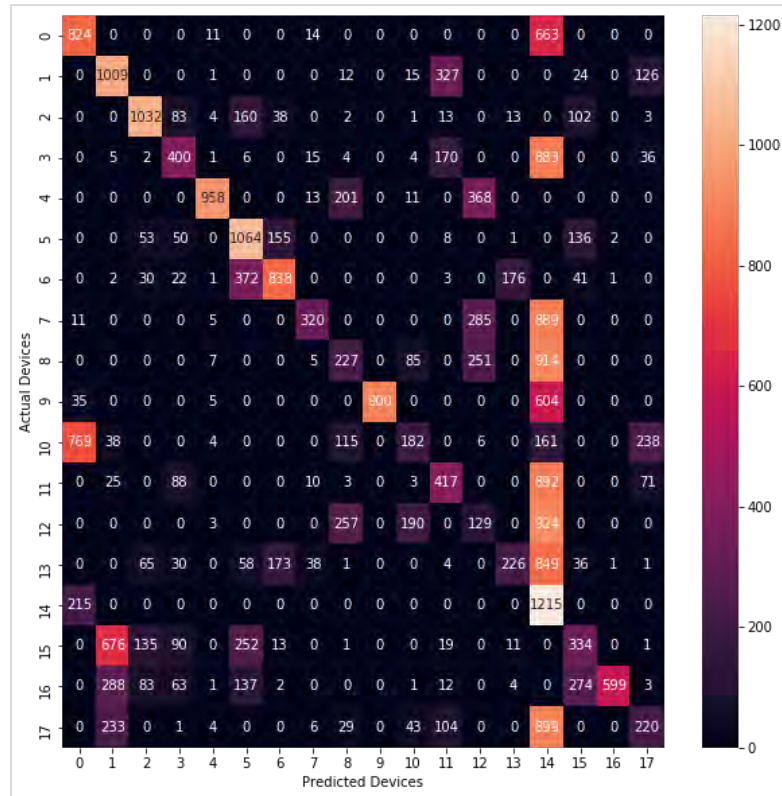


Figure 6.18: Confusion Matrix for SVM Non-Linear (Mel Input).

**ROC Curve:**

Figure 6.19 shows the ROC plots for the SVM Non-Linear Classifier. It presents the plot for each device against all. From the curve, one can interpret that the model performed badly.

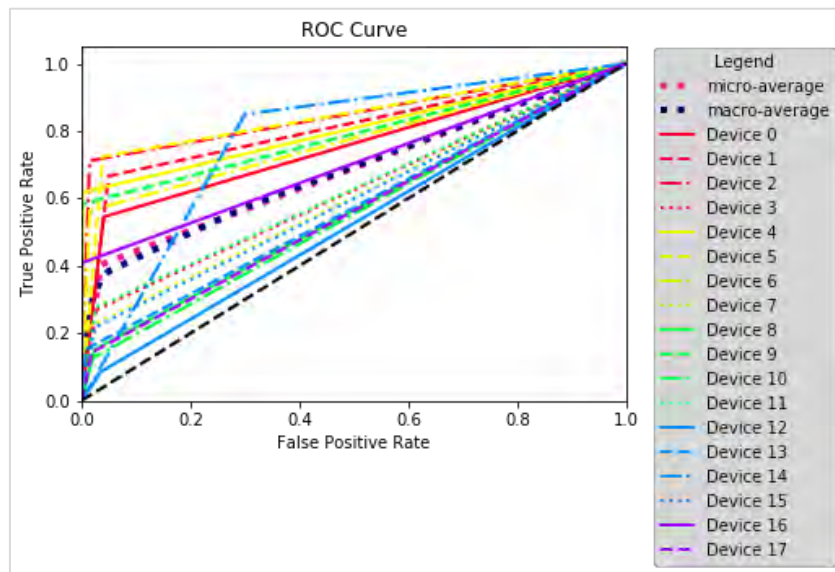


Figure 6.19: ROC Curve for SVM Non-Linear Classifier (Mel Input).

**K-Fold:**

In order to validate that the model has not been over-fitted, K-Fold is implemented using 10 folds run. The results of the 10 fold runs are shown in Table 6.35.

Table 6.35: K-Fold for SVM Non-Linear Classifier (Mel Input) - Mean (Standard Deviation)

Metric	Accuracy	Precision	Recall	F1-score
<b>Mean</b>	0.4124 (0.0012)	0.4244 (0.0013)	0.4232 (0.0012)	0.4276 (0.0011)

### Hypertuning:

In order to obtain better results, hypertuning was performed on the model. This ensures that the model has been trained using the most optimum parameters. The tuning parameters and the best result are shown in Table 6.36.

Table 6.36: SVM Non-Linear Classifier (Mel Input) Hypertuning.

Metric	Options	Best Result
<b>param_grid</b>	[0.1,1, 10,100]	0.1
<b>gamma</b>	[1,0.1,0.01,0.001]	0.01

The model was trained, and 10 fold validation was performed. The results of the mean of the 10 fold validation for each of the metrics is shown in Table 6.36. The hypertuning improved the results and increased the F1 -score from 0.42 to 0.43.

Table 6.37: K-Fold for SVM Non-Linear (Mel Input) - Mean (Standard Deviation) After Hypertuning.

Metric	Accuracy	Precision	Recall	F1-score
<b>Mean</b>	0.4354 (0.0024)	0.4356 (0.0023)	0.4343 (0.0024)	0.4367 (0.0029)

**Conclusion:** From the results, it can be interpreted that using the statistical input, the tuned model had a better result (F1-score **0.84**) in comparison with the tuned model with Mel input (F1-score **0.43**).

**6.2.4. LightGBM model.** LightGBM Model stands for Light Gradient Boosting Machine [96]. It is a Gradient Boosting framework developed by Microsoft and made opensource to the public. This model is based on decision trees and leverages two techniques:

1. Gradient-based One Side Sampling and Exclusive Feature Bundling (EFB). Gradient-based One Side Sampling is a technique that down samples the instances on the basis of the gradient [97]. Since instances with large gradient are under trained, and the ones with small gradient are well trained, LightGBM will retain the instances that have large gradient and will perform random sampling to instances that contain small gradients. This will in turn reduce the amount of data that a decision tree has to leverage for learning.
2. The second technique, EFB, is a method to reduce the number of features, by bundling very similar features into a single one [98]. This makes the model light weight and run faster in comparison with the XG-Boost or Gradient Boosts models.

**6.2.4.1. LightGBM model with statistical input.** In this section, the model with statistical input (standard deviation, sum, variance, maximum, minimum, mean, median, skewness, and kurtosis), the goodness of measures classification report of the results showing the recall, precision, and F1-Score values, the Confusion Matrix, the ROC Curve, and the K-fold results for the LightGBM Model are presented.

**Goodness of Measure Classification Report:**

Table 6.38 presents the precision, recall, and F1-score values for each of the devices. The micro, macro, and weighted average of all the classes is shown too. A macro-average is the average of all the values for each individual class. A micro-average will aggregate the contributions of all class. That is, it will compute the positive contributions (True Positives) for all classes divided by the total number of points. The results indicate a micro average score of 0.93 for all the metrics (precision, recall, F1-Score). This indicates that the model is reasonably good, however, more tuning may be required to achieve a more reliable model.

Table 6.38: LightGBM Classifier (Statistical Input) Classification Results.

Device	Precision	Recall	F1-Score
0	1	1	1
1	0.93	0.93	0.93
2	0.85	0.84	0.84
3	0.89	0.92	0.9
4	0.93	0.99	0.96
5	0.87	0.98	0.92
6	0.91	0.87	0.89
7	1	1	1
8	0.93	0.93	0.93
9	1	0.99	0.99
10	0.96	0.98	0.97
11	0.9	0.93	0.91
12	0.97	0.92	0.94
13	0.94	0.84	0.89
14	0.99	1	1
15	0.83	0.82	0.82
16	0.92	0.85	0.88
17	0.93	0.95	0.94
<b>Micro Average</b>	0.93	0.93	0.93
<b>Macro Average</b>	0.93	0.93	0.93

From the above table, one can interpret that the model performed well for almost all devices. Devices 0, 7, and 14 show the best results (F1-Score of around 1), followed by devices 9 and 10, with (F1-scores above 0.97). Following are devices 4, 8, 12, and 17 (F1-Scores between 0.93 and 0.96). The rest of devices had the models (with F1-Score 0.82 and above) which is an indication of not very accurate classification. This concludes that the model could perform well for some devices and not for others.

**Confusion Matrix:**

The confusion matrix in Figure 6.20 shows the correctly and incorrectly classified devices that the LightGBM classifier predicted. It can be shown that there are several falsely identified devices.

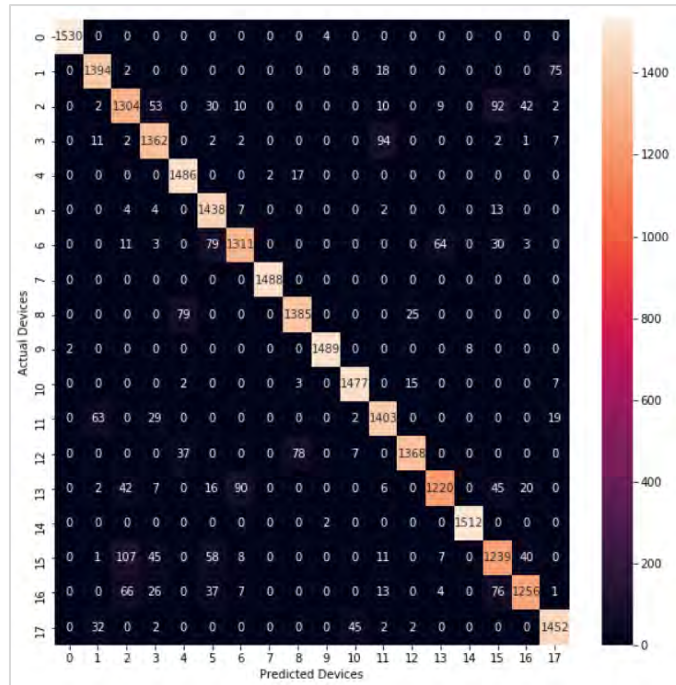


Figure 6.20: Confusion Matrix for LightGBM (Statistical Input).

**ROC Curve:**

Figure 6.21 shows the ROC plots for the LightGBM Classifier. It presents the plot for each device against all. From the curve, one can interpret that for devices 0, 7, 14, 9, and 10 the ratio of True Positive rate to False Positive Rate was the highest. A clearer interpretation of the graph is shown in Table 6.39 that shows the AUC values.

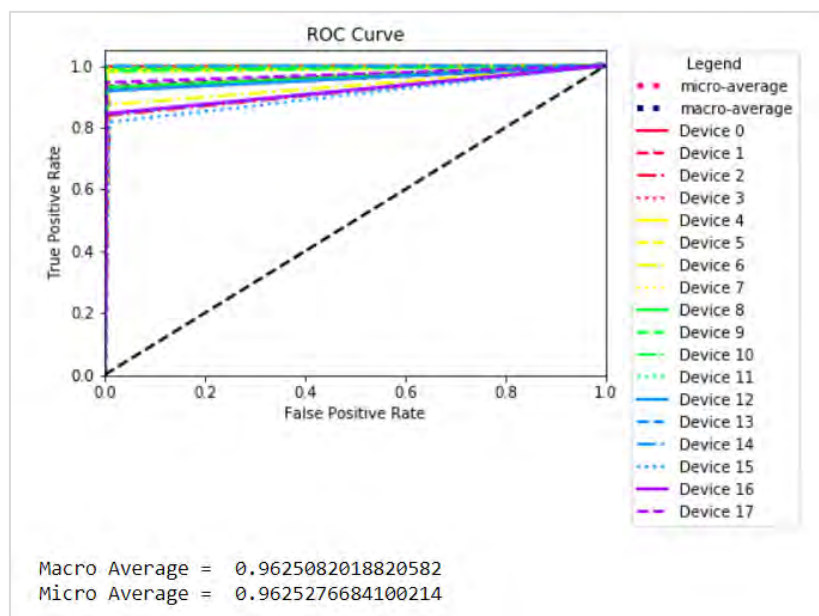


Figure 6.21: ROC Curve for LightGBM (Statistical Input).

In Table 6.39, the AUC values for each of the curves are shown. It can be shown that the highest AUC values (around 0.99) were corresponding for devices 0,7,14, 9, and 10, indicating that almost all of the devices in the sample set were correctly identified to their respective class. Devices 1, 3, 8, 11, 12 and 17 show values between (0.96 and 0.97) indicating that more than 95% devices in the sample set were correctly identified. The rest of the devices show values between (0.91 and 0.95).

Table 6.39: LightGBM Classifier (Statistical Input) AUC Values.

Device Number	AUC value
Device 0	0.9987
Device 1	0.9634
Device 2	0.9150
Device 3	0.9559
Device 4	0.9914
Device 5	0.9854
Device 6	0.9343
Device 7	1.0000
Device 8	0.9632
Device 9	0.9965
Device 10	0.9898
Device 11	0.9597
Device 12	0.9582
Device 13	0.9196
Device 14	0.9992
Device 15	0.9036
Device 16	0.9205
Device 17	0.9708
Macro Average	0.9625
Micro Average	0.9625

### **K-Fold:**

In order to validate that the model has not been over-fitted, K-Fold is implemented. The results of the 10 fold runs are shown in Table 6.40 . It can be shown that all folds resulted in an accuracy, precision, recall, and F1 -Score values are between 0.92 on average. This means that after K-fold the values match the model results, and therefore the model has not been overfitted.

Table 6.40: K-Fold for LightGBM Classifier (Statistical Input) - Mean (Standard Deviation).

Metric	Accuracy	Precision	Recall	F1-score
<b>Mean</b>	0.9241 (0.0025)	0.9248 (0.0026)	0.9241 (0.0025)	0.9238 (0.0026)

### Hypertuning:

In order to obtain better results, hypertuning was performed on the model. The tuning parameters and the best result are shown in Table 6.41.

Table 6.41: LightGBM Classifier (Statistical Input) Hypertuning.

Metric	Options	Best Result
<b>num_leaves</b>	[1, 100]	47
<b>feature_fraction</b>	[0,1]	0.8
<b>max_depth</b>	[1,100]	36
<b>num_iterations</b>	[1,100]	20
<b>max_bin</b>	[2,100]	34
<b>learning_rate</b>	[0,1]	0.004245

The model was trained, and 10 fold validation was performed. The results of the mean of the 10 fold validation for each of the metrics is shown in Table 6.42. The hypertuning improved the results and increased the F1 -score from 0.92 to 0.94.

Table 6.42: K-Fold for LightGBM (Statistical Input) - Mean (Standard Deviation) After Hypertuning.

Metric	Accuracy	Precision	Recall	F1-score
<b>Mean</b>	0.9439 (0.0023)	0.9441 (0.0023)	0.9440 (0.0023)	0.9439 (0.0023)

**6.2.4.2. LightGBM model with mel input.** In this section, the model with mel input, the goodness of measures classification report of the results showing the recall, precision, and F1-Score values, the Confusion Matrix, the ROC Curve, and the K-fold results for the LightGBM Model are presented.

### **Goodness of Measure Classification Report:**

Table 6.43 presents the precision, recall, and F1-score values for each of the devices.

Table 6.43: LightGBM Classifier (Mel Input) Classification Results.

Device	Precision	Recall	F1-Score
0	1	0.99	1
1	0.89	0.86	0.88
2	0.77	0.75	0.76
3	0.84	0.88	0.85
4	0.85	1	0.92
5	0.74	0.94	0.83
6	0.9	0.81	0.85
7	1	1	1
8	0.91	0.85	0.88
9	0.99	0.97	0.98
10	0.93	0.97	0.95
11	0.87	0.9	0.89
12	0.95	0.86	0.9
13	0.92	0.79	0.85
14	0.97	0.99	0.98
15	0.74	0.74	0.74
16	0.96	0.76	0.85
17	0.85	0.93	0.89
<b>Micro Average</b>	0.89	0.89	0.89
<b>Macro Average</b>	0.89	0.89	0.89

### **Confusion Matrix:**

The confusion matrix in Figure 6.22 shows the correctly and incorrectly classified devices that the LightGBM classifier predicted. It can be shown that there are several falsely identified devices. From the results, it can be shown that there are a huge number of devices that are incorrectly identified, indicating that the LightGBM model does not work well when the input provided is the Standard input data.

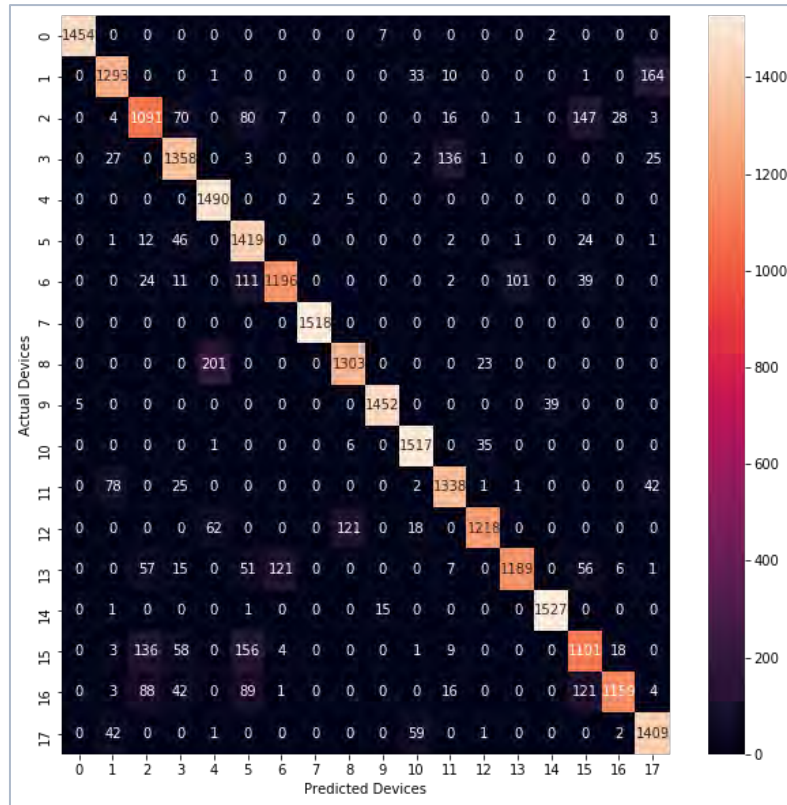


Figure 6.22: Confusion Matrix for LightGBM (Mel Input).

### ROC Curve:

Figure 6.23 shows the ROC plots for the LightGBM Classifier. It presents the plot for each device against all. From the curve, one can interpret that the model results still have a relatively high False Positive Rate.

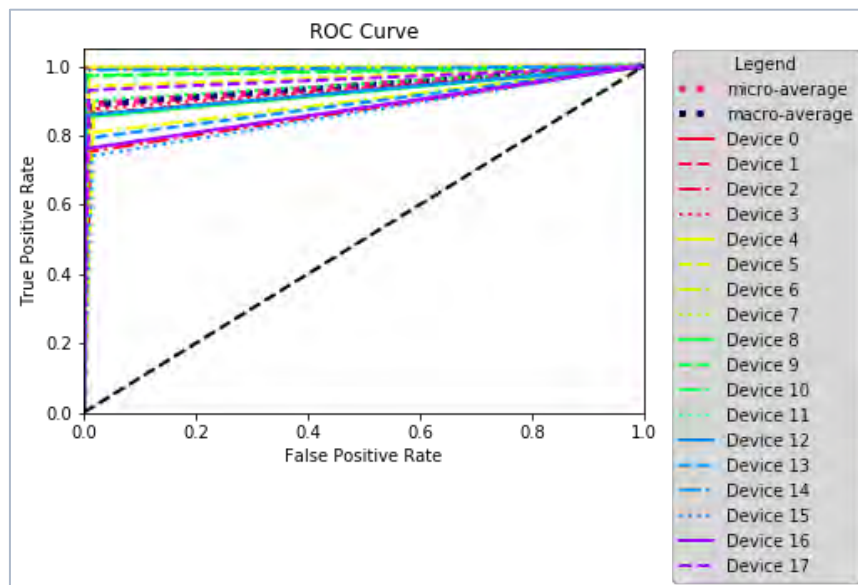


Figure 6.23: ROC Curve for LightGBM (Mel Input).

### K-Fold:

In order to validate that the model has not been over-fitted, K-Fold is implemented. The results of the 10 fold runs are shown in Table 6.44.

Table 6.44: K-Fold for LightGBM Classifier (Mel Input) - Mean (Standard Deviation).

Metric	Accuracy	Precision	Recall	F1-score
<b>Mean</b>	0.8877 (0.0038)	0.8924 (0.0031)	0.8879 (0.0038)	0.8876 (0.0037)

### Hypertuning:

In order to obtain better results, hypertuning was performed on the model. The tuning parameters and the best result are shown in Table 6.45.

Table 6.45: LightGBM Classifier (Mel Input) Hypertuning.

Metric	Options	Best Result
<b>num_leaves</b>	[1, 100]	32
<b>feature_fraction</b>	[0,1]	0.65
<b>max_depth</b>	[1,100]	32
<b>num_iterations</b>	[1,100]	29
<b>max_bin</b>	[2,100]	65
<b>learning_rate</b>	[0,1]	0.0065349

The model was trained, and 10 fold validation was performed. The results of the mean of the 10 fold validation for each of the metrics is shown in Table 6.46. The hypertuning improved the results and increased the F1 -score from 0.88 to 0.89.

Table 6.46: K-Fold for LightGBM (Mel Input) - Mean (Standard Deviation) After Hypertuning.

Metric	Accuracy	Precision	Recall	F1-score
<b>Mean</b>	0.8912 (0.0041)	0.8949 (0.0036)	0.8914 (0.0041)	0.8910 (0.0041)

**Conclusion:** From the results, it can be interpreted that using the statistical input, the tuned model had a better result (F1-score **0.94**) in comparison with the tuned model with Mel input (F1-score **0.89**).

**6.2.5. Gradient boost model.** Gradient Boost is a machine learning model used in classification and regression problems. The algorithm is based on three steps. Optimization of a loss function, construction of decision trees in a greedy manner (choosing the best split points based on purity values such as the Gini score or to minimizing the loss), and finally the Trees are added one at a time [99]. A gradient descent procedure (such as changing weights) is used to minimize the loss when adding the trees. An example for the Gradient Boost algorithm is shown in Figure 6.24. It can be shown how the decision trees are constructed in every iteration given the X and Y values.

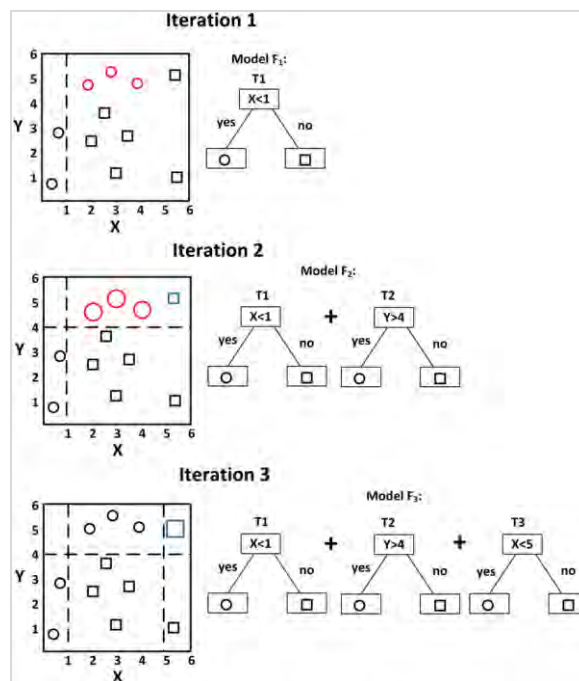


Figure 6.24: Gradient Boost Algorithm Example [100].

**6.2.5.1. Gradient boost with statistical input.** This section presents the model built using the statistical input for the Gradient Boost classifier. The experiment parameters are shown in Table 6.47. The default hyperparameters were used to train the model. Also, the goodness of measures classification report of the results showing the recall, precision, and F1-Score values, the Confusion Matrix, the ROC Curve, and the K-fold results for the Gradient Boost Model are presented.

**Experiment parameters:**

Table 6.47: Gradient Boost (Statistical Input) Classifier Experiment Parameters.

Input parameters	Values
<b>max_depth</b>	5
<b>learning_rate</b>	0.1
<b>n_estimators</b>	10
<b>Training and Testingsplits</b>	Training: 70% Testing: 30%

**Goodness of Measure Classification Report:**

Table 6.48 presents the precision, recall, and F1-score values for each of the devices. The micro, macro, and weighted average of all the classes is shown too. A macro-average is the average of all the values for each individual class. A micro-average will aggregate the contributions of all class. That is, it will compute the positive contributions (True Positives) for all classes divided by the total number of points.

Table 6.48: Gradient Boost Classifier (Statistical Input) Classification Results.

Device	Precision	Recall	F1-Score
<b>0</b>	1	1	1
<b>1</b>	0.96	0.96	0.96
<b>2</b>	0.9	0.78	0.84
<b>3</b>	0.69	0.93	0.79
<b>4</b>	0.87	0.9	0.89
<b>5</b>	0.98	0.85	0.91
<b>6</b>	0.96	0.96	0.96
<b>7</b>	0.88	0.8	0.84
<b>8</b>	0.94	0.98	0.96
<b>9</b>	1	1	1
<b>10</b>	0.8	0.73	0.76
<b>11</b>	0.84	0.87	0.86
<b>12</b>	0.9	0.88	0.89
<b>13</b>	0.8	0.96	0.87
<b>14</b>	0.85	1	0.92
<b>15</b>	0.92	0.8	0.86
<b>16</b>	0.73	0.76	0.74
<b>17</b>	1	0.73	0.84
<b>Micro Average</b>	0.88	0.88	0.88
<b>Macro Average</b>	0.89	0.88	0.88

From the above table, one can interpret that the model did not perform well for most of the devices. Devices 0 and 9 show the highest results (F1-Score of around 1), followed by devices 5 and 14, with (F1-scores above 0.9). The rest of the devices show (F1-Score less than 0.9) Following are devices 4, 8, 12, and 17 (F1-Scores between 0.93 and 0.96). The rest of devices. This concludes that the model could perform well for only for few numbers of devices and not for others, and therefore is not a good model.

**Confusion Matrix:**

The confusion matrix in Figure 6.25 shows the correctly and incorrectly classified devices that the Gradient Boost classifier predicted. It can be shown that there are many devices have been falsely classified. A clearer interpretation can be shown in the ROC and AUC values next.

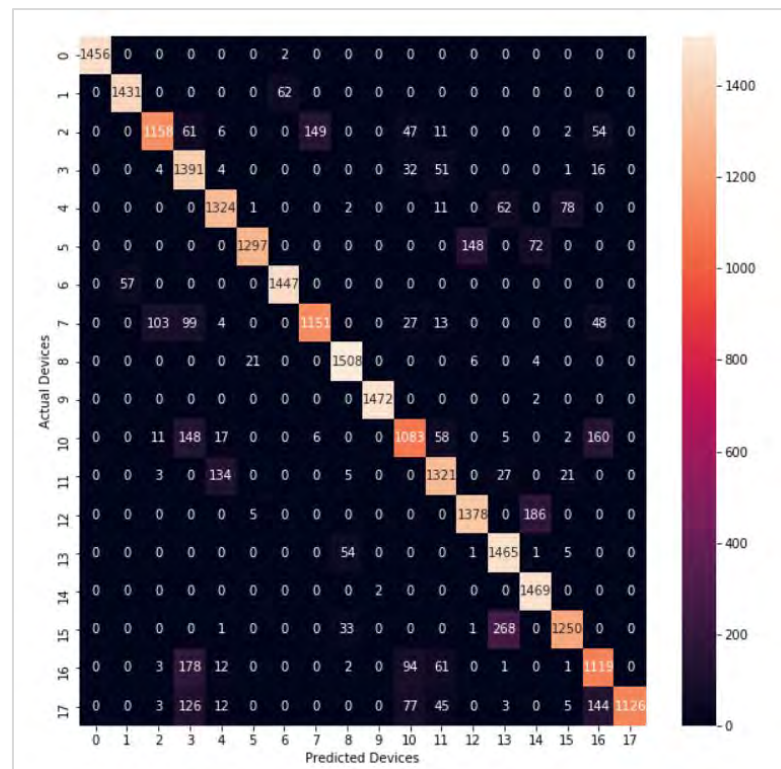


Figure 6.25: Confusion Matrix for Gradient Boost (Statistical Input).

**ROC Curve:**

Figure 6.26 shows the ROC plots for the Gradient Boost Classifier. It presents the plot for each device against all. From the curve, one can interpret that for devices 0, 9, 5, and 14 the ratio of True Positive rate to False Positive Rate was the highest. A clearer interpretation of the graph is shown in Table 6.49 that shows the AUC values.

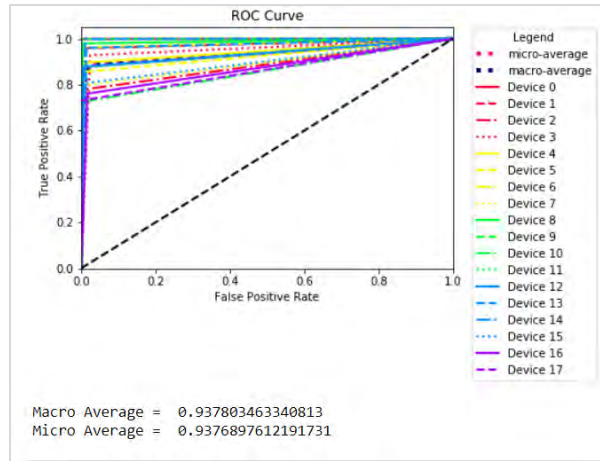


Figure 6.26: ROC Curve for Gradient Boost Classifier (Statistical Input).

In Table 6.49, the AUC values for each of the curves are shown. It can be shown that the highest AUC values (around 0.99) were corresponding for devices 0, 9, and 14, indicating that almost all of the devices in the sample set were correctly identified to their respective class. The micro average result is 0.93.

Table 6.49: Gradient Boost Classifier AUC Values (Statistical Input).

Device Number	AUC value
Device 0	0.9993
Device 1	0.9781
Device 2	0.8866
Device 3	0.9520
Device 4	0.9442
Device 5	0.9270
Device 6	0.9798
Device 7	0.8952
Device 8	0.9880
Device 9	0.9993
Device 10	0.8580
Device 11	0.9322
Device 12	0.9361
Device 13	0.9728
Device 14	0.9941
Device 15	0.9002
Device 16	0.8721
Device 17	0.8653
Micro Average	0.9376

### K-Fold:

In order to validate that the model has not been over-fitted, K-Fold is implemented. The results of the 10 fold runs are shown in Table 6.50 . It can be shown that all folds resulted in an accuracy, precision, recall, and F1 -Score values are close to 0.88 as shown in average. This means that after K-fold the values match the model results, and therefore the model has not been overfitted.

Table 6.50: K-Fold for Gradient Boost (Statistical Input) - Mean (Standard Deviation).

Metric	Accuracy	Precision	Recall	F1-score
<b>Mean</b>	0.8819 (0.0026)	0.8898 (0.0036)	0.8818 (0.0027)	0.8819 (0.0027)

### Hypertuning:

In order to obtain better results, hypertuning was performed on the model. The tuning parameters and the best result are shown in Table 6.51.

Table 6.51: Gradient Boost Classifier (Statistical Input) Hypertuning.

Metric	Options	Best Result
<b>min_samples_split</b>	[0,10]	1
<b>min_samples_leaf</b>	[0,10]	3
<b>min_weight_fraction_leaf</b>	[0,1]	0
<b>max_depth</b>	[1,50]	5
<b>max_features</b>	['auto', 'sqrt', 'log2']	auto
<b>learning_rate</b>	[0,1]	0.0003843
<b>n_estimators</b>	[1,100]	47

The model was trained, and 10 fold validation was performed. The results of the mean of the 10 fold validation for each of the metrics is shown in Table 6.52. The hypertuning improved the results and increased the F1 -score from 0.88 to 0.90.

Table 6.52: K-Fold for Gradient Boost (Statistical Input) - Mean (Standard Deviation) After Hypertuning.

Metric	Accuracy	Precision	Recall	F1-score
Mean	0.8900 (0.0022)	0.9022 (0.0022)	0.9065 (0.0021)	0.9024 (0.0021)

**6.2.5.2. Gradient boost with mel input.** This section presents the model built using the mel input for the Gradient Boost classifier. The experiment parameters are shown in Table 6.53. The default hyperparameters were used to train the model. Also, the goodness of measures classification report of the results showing the recall, precision, and F1-Score values, the Confusion Matrix, the ROC Curve, and the K-fold results for the Gradient Boost Model are presented.

**Experiment parameters:**

Table 6.53: Gradient Boost (Mel Input) Classifier Experiment Parameters.

Input parameters	Values
max_depth	5
learning_rate	0.1
n_estimators	10
Training and Testing splits	Training: 70% Testing: 30%

**Goodness of Measure Classification Report:**

Table 6.54 presents the precision, recall, and F1-score values for each of the devices. The micro, macro, and weighted average of all the classes is shown too. A macro-average is the average of all the values for each individual class. A micro-average will aggregate the contributions of all class. That is, it will compute the positive contributions (True Positives) for all classes divided by the total number of points. For the Gradient Boost Classifier, the results show a micro average score of 0.93. This indicates that the model will need to be further tuned to provide more reliable results.

Table 6.54: Gradient Boost Classifier (Mel Input) Classification Results.

Device	Precision	Recall	F1-Score
0	0.82	0.84	0.83
1	0.98	0.99	0.99
2	0.91	0.95	0.93
3	0.93	0.91	0.92
4	0.99	0.98	0.99
5	0.93	0.91	0.92
6	0.93	0.98	0.95
7	0.93	0.86	0.89
8	0.92	0.92	0.92
9	0.92	0.93	0.92
10	1	1	1
11	0.88	0.96	0.92
12	0.84	0.84	0.84
13	0.96	0.98	0.97
14	0.91	0.88	0.89
15	1	1	1
16	0.96	0.93	0.95
17	0.93	0.87	0.9
<b>Micro Average</b>	0.93	0.93	0.93
<b>Macro Average</b>	0.93	0.93	0.93

### Confusion Matrix:

The confusion matrix in Figure 6.27 shows the correctly and incorrectly classified devices that the Gradient Boost classifier predicted. It can be shown that there are many devices that have been falsely classified. A clearer interpretation can be shown in the ROC and AUC values next.

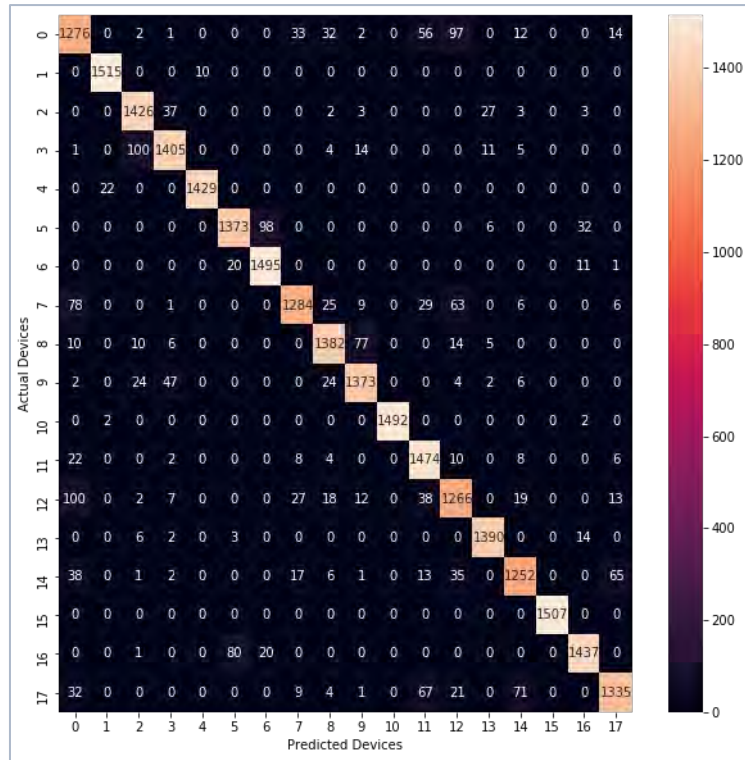


Figure 6.27: Confusion Matrix for Gradient Boost (Mel Input).

**ROC Curve:**

Figure 6.28 shows the ROC plots for the Gradient Boost Classifier. It presents the plot for each device against all.

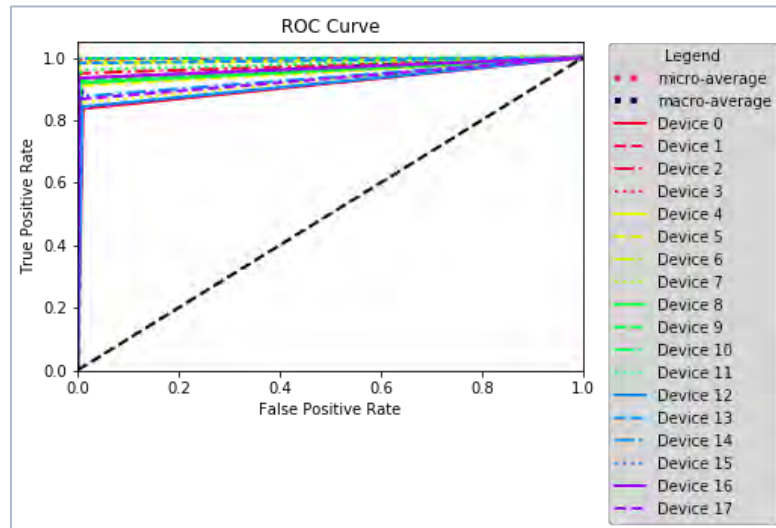


Figure 6.28: ROC Curve for Gradient Boost Classifier (Mel Input).

**K-Fold:**

In order to validate that the model has not been over-fitted, K-Fold is implemented. The results of the 10 fold runs are shown in Table 6.55. It can be shown

that all folds resulted in an accuracy, precision, recall, and F1-Score values are close to 0.88 as shown in average. This means that after K-fold the values match the model results, and therefore the model has not been overfitted.

Table 6.55: K-Fold for Gradient Boost (Mel Input) - Mean (Standard Deviation).

Metric	Accuracy	Precision	Recall	F1-score
<b>Mean</b>	0.9238 (0.0026)	0.9374 (0.0036)	0.9384 (0.0027)	0.9342 (0.0027)

### Hypertuning:

In order to obtain better results, hypertuning was performed on the model. The tuning parameters and the best result are shown in Table 6.56.

Table 6.56: Gradient Boost Classifier (Mel Input) Hypertuning.

Metric	Options	Best Result
<b>min_samples_split</b>	[0,10]	3
<b>min_samples_leaf</b>	[0,10]	7
<b>min_weight_fraction_leaf</b>	[0,1]	0
<b>max_depth</b>	[1,50]	10
<b>max_features</b>	['auto', 'sqrt', 'log2']	auto
<b>learning_rate</b>	[0,1]	0.0005432
<b>n_estimators</b>	[1,100]	55

The model was trained, and 10 fold validation was performed. The results of the mean of the 10 fold validation for each of the metrics is shown in Table 6.57. The hypertuning improved the results and increased the F1 -score from 0.93 to 0.94.

Table 6.57: K-Fold for Gradient Boost (Mel Input) - Mean (Standard Deviation) After Hypertuning.

Metric	Accuracy	Precision	Recall	F1-score
<b>Mean</b>	0.9409 (0.0024)	0.9462 (0.0023)	0.9498 (0.0024)	0.9443 (0.0024)

**Conclusion:** From the results, it can be interpreted that using the mel input, the tuned model had a better result (F1-score **0.94**) in comparison with the tuned model with statistical input (F1-score **0.90**).

**6.2.6. XG-Boost model.** XG-Boost also known as Extreme Gradient Boosting is one of the gradient boosting techniques that provide better performance and speed in the tree-base machine learning algorithms. It is different from the Gradient Boosting in that XG-Boost it follows a more regulated model formalization to reduce over-fitting. XG-Boost has added capabilities in that it allows for parallelization where trees can be constructed over multiple CPU cores and caches the data structures [101].

**6.2.6.1. XG-Boost with statistical input.** In this section, the model built using the statistical input, the goodness of measures classification report of the results showing the recall, precision, and F1-Score values, the Confusion Matrix, the ROC Curve, and the K-fold results for the XG-Boost Model are presented. The experiment parameters for the XG-Boost classifier are shown in Table 6.58. The default hyperparameters were used to train the model.

**Experiment parameters:**

Table 6.58: XG-Boost Classifier Experiment Parameters.

Input parameters	Values
colsample_bytree	0.3
alpha	10
max_depth	5
learning_rate	0.1
n_estimators	10
Training and Testingsplits	Training: 70% Testing: 30%

**Goodness of Measure Classification Report:**

Table 6.59 presents the precision, recall, and F1-score values for each of the devices. The micro, macro, and weighted average of all the classes is shown too. A macro-average is the average of all the values for each individual class. A micro-average will aggregate the contributions of all class. That is, it will compute the positive contributions (True Positives) for all classes divided by the total number of points.

Table 6.59: XG-Boost Classifier Classification Results (Statistical Input).

Device	Precision	Recall	F1-Score
0	0.84	0.87	0.85
1	0.89	0.66	0.75
2	0.86	0.87	0.86
3	0.87	0.96	0.91
4	0.73	0.87	0.79
5	0.8	0.84	0.82
6	0.98	1	0.99
7	0.86	0.88	0.87
8	0.9	0.88	0.89
9	0.92	0.97	0.95
10	0.85	0.89	0.87
11	0.97	0.85	0.91
12	0.76	0.72	0.74
13	0.88	0.9	0.89
14	0.67	0.77	0.71
15	1	0.73	0.84
16	0.86	0.87	0.86
17	0.84	0.87	0.85
<b>Micro Average</b>	0.86	0.86	0.86
<b>Macro Average</b>	0.87	0.86	0.86

From the above table, one can interpret that the model didn't perform well for most of the devices. Only for device 6 it presents a high value (F1-Score of around 0.99), followed by devices 3 and 11, with (F1-scores above 0.9). The rest of the devices show (F1-Score between 0.7 and 0.89). This concludes that the model could perform for most of the devices and therefore is not a good model.

### Confusion Matrix:

The confusion matrix in Figure 6.29 shows the correctly and incorrectly classified devices that the Gradient Boost classifier predicted. It can be shown that there are many devices have been falsely classified. A clearer interpretation can be shown in the ROC and AUC values next.

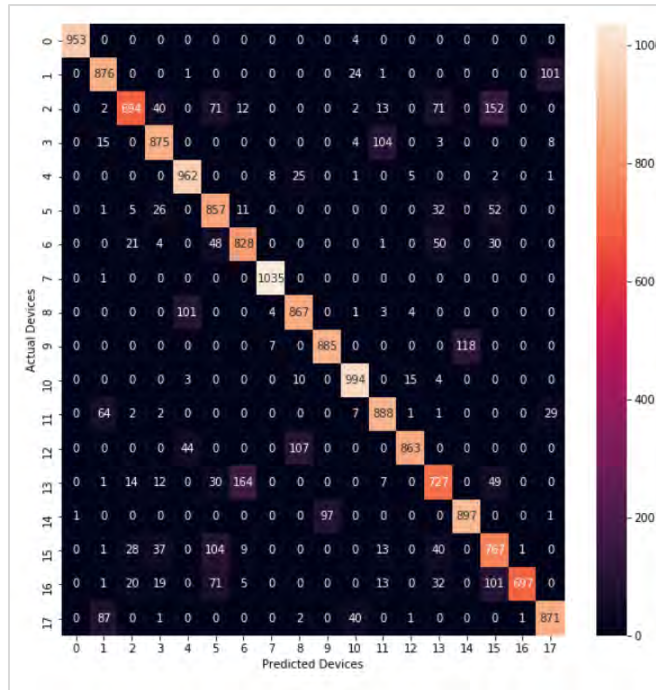


Figure 6.29: Confusion Matrix for XG-Boost (Statistical Input).

**ROC Curve:**

Figure 6.30 shows the ROC plots for the XG-Boost Classifier. It presents the plot for each device against all. From the curve, one can interpret that only for few devices the ROC shows a good plot, while most of them had a little bit low True Positive to False Positive rate. A clearer interpretation of the graph is shown in Table 6.60 that shows the AUC values and the exact meaning of these plots.

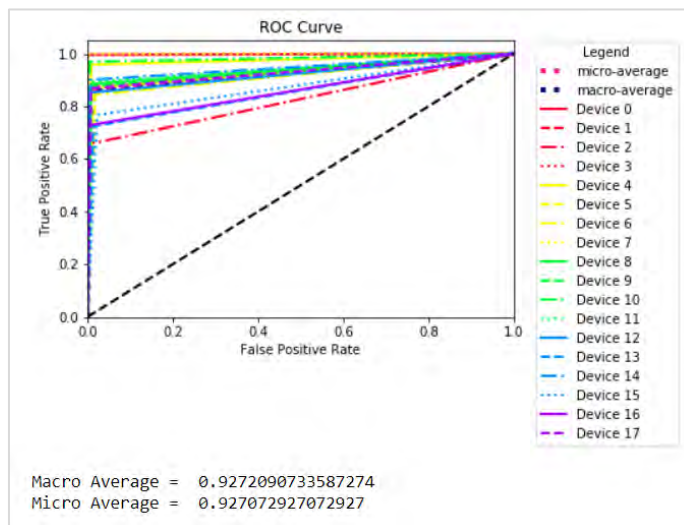


Figure 6.30: ROC Curve for XG-Boost Classifier (Statistical Input).

In Table 6.60, the AUC values for each of the curves are shown. It can be shown that the highest AUC values (around 0.99) were corresponding for devices 0 and 7, indicating that almost all of the devices in the sample set were correctly identified to their respective class. Devices 10 show value of (around 0.98) indicating that more than 98% devices in the sample set were correctly identified. The rest of the devices show values of AUC 0.95 indicating more False Positive results. In the next section we present the K-fold implementation of the model to check for over-fitting.

Table 6.60: XG-Boost Classifier AUC Values (Statistical Input).

Device Number	AUC value
Device 0	0.9979
Device 1	0.9316
Device 2	0.8256
Device 3	0.9295
Device 4	0.9747
Device 5	0.9260
Device 6	0.9157
Device 7	0.9990
Device 8	0.9381
Device 9	0.9353
Device 10	0.9820
Device 11	0.9421
Device 12	0.9248
Device 13	0.8552
Device 14	0.9468
Device 15	0.8722
Device 16	0.8633
Device 17	0.9301
Macro Average	0.9272
Micro Average	0.9270

### **K-Fold:**

In order to validate that the model has not been over-fitted, K-Fold is implemented. The results of the 10 fold runs are shown in Table 6.61. It can be shown that all folds resulted in an accuracy, precision, recall, and F1-Score values are close to

0.88 on average. This means that after K-fold the values match the model results, and therefore the model has not been overfitted.

Table 6.61: K-Fold (Statistical Input) for XG-Boost Classifier - Mean (Standard Deviation).

Metric	Accuracy	Precision	Recall	F1-score
<b>Mean</b>	0.8652 (0.0050)	0.8713 (0.0047)	0.8651 (0.0050)	0.8650 (0.0051)

### Hypertuning:

In order to obtain better results, hypertuning was performed on the model. The tuning parameters and the best result are shown in Table 6.62.

Table 6.62: XG-Boost Classifier (Statistical Input) Hypertuning.

Metric	Options	Best Result
<b>min_samples_split</b>	[0,10]	2
<b>min_samples_leaf</b>	[0,10]	2
<b>min_weight_fraction_leaf</b>	[0,1]	0
<b>max_depth</b>	[1,50]	10
<b>max_features</b>	['auto', 'sqrt', 'log2']	auto
<b>learning_rate</b>	[0,1]	0.00012
<b>n_estimators</b>	[1,100]	30

The model was trained, and 10 fold validation was performed. The results of the mean of the 10 fold validation for each of the metrics is shown in Table 6.63. The hypertuning improved the results and increased the F1 -score from 0.86 to 0.87.

Table 6.63: K-Fold for XG-Boost (Statistical Input) - Mean (Standard Deviation) After Hypertuning.

Metric	Accuracy	Precision	Recall	F1-score
<b>Mean</b>	0.8781 (0.0033)	0.8843 (0.0027)	0.8780 (0.0033)	0.8781 (0.0033)

**6.2.6.2. XG-Boost with mel input.** In this section, the model built using the mel input, recall, precision, and F1-Score values, the Confusion Matrix, the ROC, and the K-fold results for the XG-Boost Model are presented. The experiment parameters are shown in Table 6.64. The default hyperparameters were used to train the model.

**Experiment parameters:**

Table 6.64: XG-Boost Classifier Experiment Parameters.

Input parameters	Values
<b>colsample_bytree</b>	0.3
<b>alpha</b>	10
<b>max_depth</b>	5
<b>learning_rate</b>	0.1
<b>n_estimators</b>	10
<b>Training and Testingsplits</b>	Training: 70% Testing: 30%

Table 6.65 presents the results for each of the devices.

Table 6.65: XG-Boost Classifier (MelInput) Classification Results.

Device	Precision	Recall	F1-Score
0	0.56	0.4	0.46
1	0.96	0.59	0.73
2	0.36	0.72	0.48
3	0.67	0.67	0.67
4	0.52	0.58	0.54
5	0.83	0.61	0.71
6	0.5	0.38	0.43
7	0.55	0.79	0.65
8	0.77	0.59	0.67
9	0.93	0.54	0.69
10	0.35	0.6	0.45
11	0.6	0.48	0.54
12	0.94	0.6	0.73
13	0.46	0.71	0.56
14	0.58	0.92	0.71
15	0.91	0.6	0.72
16	0.98	0.7	0.81
17	0.85	0.58	0.69
<b>Micro Average</b>	0.61	0.61	0.61

From the above table, one can interpret that the model didn't perform well for most of the devices.

### Confusion Matrix:

The confusion matrix in Figure 6.31 shows the correctly and incorrectly classified devices that the Gradient Boost classifier predicted. It can be shown that there are many devices have been falsely classified.

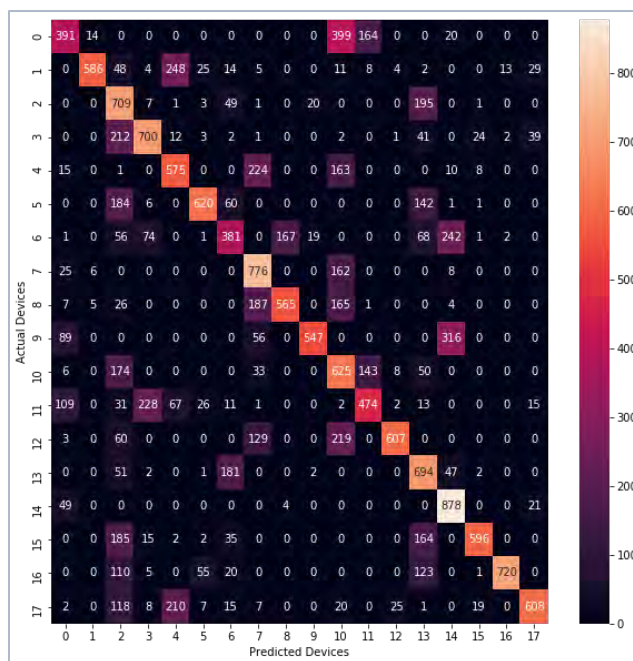


Figure 6.31: Confusion Matrix for XG-Boost (Mel Input).

### ROC Curve:

Figure 6.32 shows the ROC plots for the XG-Boost Classifier. It presents the plot for each device against all.

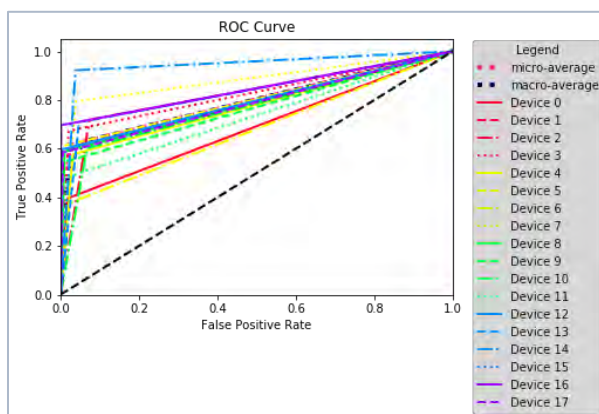


Figure 6.32: ROC Curve for XG-Boost Classifier (Mel Input).

### K-Fold:

In order to validate that the model has not been over-fitted, K-Fold is implemented. The results of the 10 fold runs are shown in Table 6.66.

Table 6.66: K-Fold (MelInput) for XG-Boost Classifier - Mean (Standard Deviation).

Metric	Accuracy	Precision	Recall	F1-score
<b>Mean</b>	0.6091 (0.0066)	0.6918 (0.0082)	0.6089 (0.0066)	0.6214 (0.0044)

### Hypertuning:

In order to obtain better results, hypertuning was performed on the model. The tuning parameters and the best result are shown in Table 6.62.

Table 6.67: XG-Boost Classifier (MelInput) Hypertuning.

Metric	Options	Best Result
<b>min_samples_split</b>	[0,10]	3
<b>min_samples_leaf</b>	[0,10]	1
<b>min_weight_fraction_leaf</b>	[0,1]	0
<b>max_depth</b>	[1,50]	10
<b>max_features</b>	['auto', 'sqrt', 'log2']	auto
<b>learning_rate</b>	[0,1]	0.0023
<b>n_estimators</b>	[1,100]	44

The model was trained, and 10 fold validation was performed. The results of the mean of the 10 fold validation for each of the metrics is shown in Table 6.68. The hypertuning improved the results and increased the F1 -score from 0.62 to 0.64.

Table 6.68: K-Fold for Gradient Boost (Mel Input) - Mean (Standard Deviation) After Hypertuning.

Metric	Accuracy	Precision	Recall	F1-score
<b>Mean</b>	0.6444 (0.0034)	0.6343 (0.0042)	0.6448 (0.0032)	0.6467 (0.0034)

**Conclusion:** From the results, it can be interpreted that using the statistical input, the tuned model had a better result (F1-score **0.87**) in comparison with the tuned model with mel input (F1-score **0.64**).

**Summary of Results:**

As a conclusion, the summary of all F1-Scores for the 10 fold validation for each of the models is shown. For the Bayesian model, hypertuning was not performed as both models with both inputs resulted in very weak results, and no work on them has been taken forward. It can be concluded that for all models except the Gradient Boost and Bayesian, when the input was the statistical input, better results were achieved. It is also concluded that, the CNN still outperforms the traditional machine learning models when they are provided with the same input also.

Table 6.69: F1-Score Results for 10-Folds Runs for all Models – (Mean and Standard Deviation).

Model	Model with Stat. Input	Tuned Model with Stat. Input	Model with Mel Input	Tuned Model with Mel Input
<b>Bayesian</b>	0.1683 (0.0016)	Not performed	0.2169 (0.0024)	Not performed
<b>SVM Linear</b>	0.4546 (0.0035)	0.5343 (0.0040)	0.2169 (0.0024)	0.3232 (0.0004)
<b>SVM Non-Linear</b>	0.8221 (0.0031)	0.8421 (0.0021)	0.4276 (0.0011)	0.4367 (0.0029)
<b>XG-Boost</b>	0.8650 (0.0051)	0.8781 (0.0033)	0.6214 (0.0044)	0.6467 (0.0034)
<b>Gradient Boost</b>	0.8819 (0.0027)	0.9024 (0.0021)	0.9342 (0.0027)	0.9443 (0.0024)
<b>LightGBM</b>	0.9238 (0.0026)	0.9439 (0.0023)	0.8876(0.0037)	0.8910 (0.0041)
<b>Random Forest</b>	0.9501 (0.0036)	0.9673 (0.0038)	0.9044 (0.0049)	0.9356 (0.0051)

## Chapter 7. Convolutional Neural Networks: Experiment 1

In this chapter, the thesis experiment in involving how Convolutional Neural Networks can be leveraged to uniquely identify a device is presented. In this experiment, only a single feature was leveraged (Inter-Arrival Time) to train a CNN and accurately identify a device. The CNN outperformed the traditional models described in Chapter 6. This chapter is divided as follows. In section 7.1, we describe what CNNs are. In section 7.2, we describe how input of the CNN model pre-processing was done, and how they were converted using Mel Specs before training the model. Section 7.3 presents what the CNN architecture is, associated filters, layers, and outputs. Section 7.4 describes how hypertuning was performed to get the most optimum parameters for the model and the results of it. Section 7.5 explains the model, and presenting the model features, such as the feature maps, model filters and their output etc. In section 7.6, the chapter is concluded by presenting the metrics performed to test for goodness of model (over-fitting) is shown along with its results.

### 7.1. Convolutional Neural Networks

Convolutional Neural Networks is a Deep Learning algorithm that is primarily used in image classification. It has also been widely used in other classification problems.

CNN consists of different layers type:

- Convolutional Layer,
- Pooling Layer, and
- Fully-Connected Layer.

**Convolutional Layer:** The Convolutional Layer consists of a set of learnable filters. A filter is similar to a matrix with width and height. If the input is having a depth of 3, then a filter will contain width, height, and depth. If the input is only a single channel, then the depth is 1. Common filter sizes are 1 by 1, 3 by 3, 5 by 5. The filter will slide across the input, with a value equal to the *stride* [102]. A stride specifies how much movement a filter must move within each iteration. For example, when the value of stride is 1, then filter moves one pixel at a time across the entire input. The result is a dot product between the inputs and filter. An example of a Convolutional Layer is illustrated in Figure 7.1 .

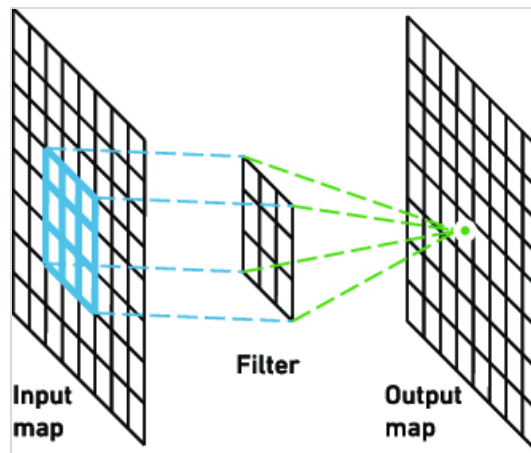


Figure 7.1 Convolutional Layer [102].

**Pooling Layer:** Pooling layer is a layer generally present after the convolutional layer to down sample the feature maps. It performs this by summarizing the features in the feature maps [103]. To accomplish this, there are two types of pooling layers: Average Pooling and Max Pooling. The Average Pooling layer will get the average for each patch of the feature map. The Max Pooling layer will get the maximum for each patch of the feature map. An illustration of Max and Averaging Pooling is shown in Figure 7.2.

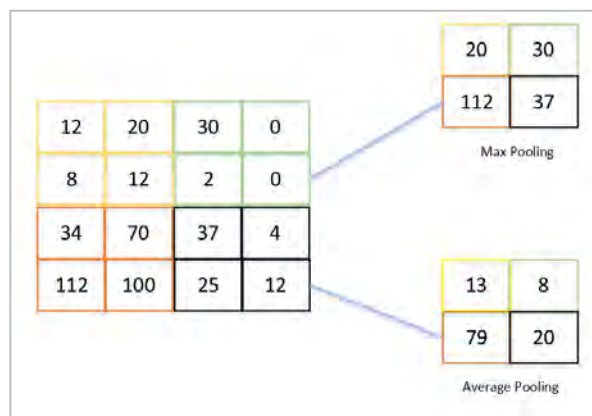


Figure 7.2: Max and Average Pooling.

**Fully-Connected Layer:** A fully connected layer is generally present in the last layers of the Neural Network and is referred to as fully-connected since all inputs from one layer are connected to every unit in the next layer [104]. It takes the output of the convolutional or pooling layer and classify the input to its corresponding label. An example of a fully-connected layer that performs a classification is shown in Figure 7.3.

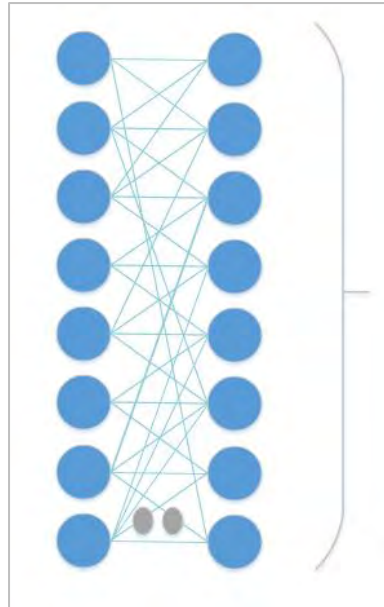


Figure 7.3: Fully-Connected Layer.

Putting these layers together, Figure 7.4 shows an example of a CNN with input size 48 x 48, the convolutional, Pooling and Fully Connected layers.

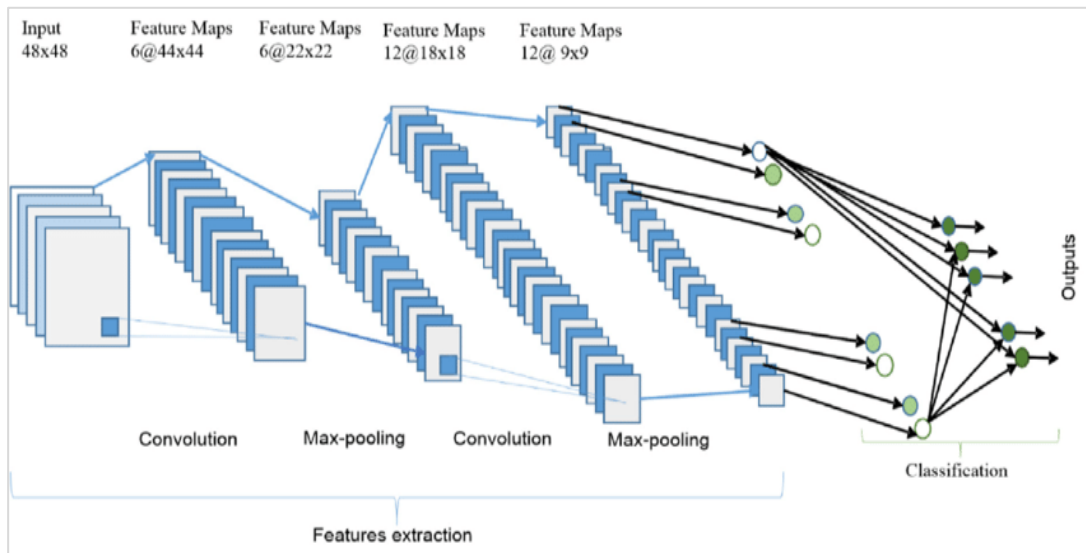


Figure 7.4: CNN Architecture.

## 7.2. Input to the Convolutional Neural Network

In this section, we present the input of the CNN. The data that has been generated and collected from the ESP32 devices (Inter-Arrival Time) are considered a sequential time series data. This data is time series signal and be considered as an audio signal. That is, the processing commonly done on audio would apply similarly to the generated data from the devices. For that, a widely used procedure in speech recognition has been applied to the data signal, Mel Frequency Cepstral Coefficients (MFCC)

[105]. The output of this procedure is the MFCC and is the input to the CNN. This procedure consists of multiple steps and are described step by step.

In audio processing, MFCC is used to for providing a better representation of sound. They perform this by generating a Mel scale that will relate the perceived frequency to its actual measured frequency. It creates Mel filterbanks that consist of multiple filters. These filters are narrow at low frequencies and get wider as the frequencies increase [106]. These banks give an idea of how much energy exists in the various energies in the signal. The overall steps to convert to MFCC are:

1. Sampling the signal. The signal from our generated data (IAT) has been sampled to a Sampling Rate of **10,000 Hz**.
2. Framing the signal. This involves framing our generated data (IAT) into very small windows of **50 milliseconds** each.
3. **Fast Fourier Transform** is applied to each of the frames, to provide the spectral components and frequency information in each frame.
4. **Periodogram** is calculated for each of the resultant frames, to identify which frequencies are present in the frame.
5. Compute the **Mel Filter Banks**.
6. **Apply** the Mel Filter Banks to the Periodograms.
7. **Sum up** the above results to get an idea of how much energy exists in various frequency regions.
8. Take the **log** of the above result.
9. Take the **Discrete Cosine Transform** of the above log-ed results, to remove correlation between the filter bank energies.
10. **Use** the results for specific period of time of the signal as input to the neural network.

The purpose of applying this procedure and converting the data is to be able to visually identify if the Inter-Arrival Time feature can be leveraged to uniquely distinguish between the devices. If the MFCC plots were (visually) different, across the time, this means that this feature can be a possible candidate to be used to distinguish between the devices. The above steps are presented in detail in Appendix A.

**Parameters used:**

- **Fs (Sampling Frequency):** 10,000 Hz
- **nfft:** 512
- **windowSize:** 512 (50 ms)

**Mel Parameters:**

- **Lowmelfreq:**  $1125 * \log(1 + (50/700)) - 50$  is the low frequency in Hz
- **Highmelfreq:**  $1125 * \log(1 + (5000/700)) - 5000$  is the high frequency in Hz
- **noOfBanks:** 23
- **mininverse:**  $700 * (\exp(\text{mspace}/1125) - 1)$

The Mel Filter banks is shown in Figure 7.5.

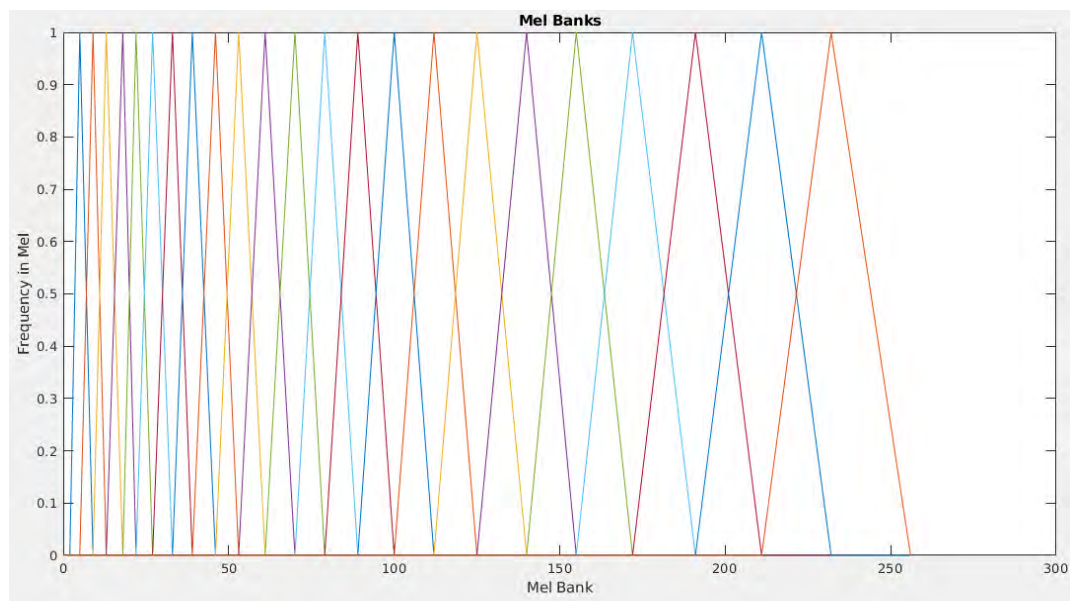


Figure 7.5: MelBanks Plot.

In order to visually see the MFCC, Figure 7.6 show a 1 second interval for a device is presented. The plots for the rest of the 17 devices is shown in Appendix B.

This includes:

- The time signal plot
- The log of the Mel Filter bank energies (step 4 above)
- The Mel frequency spectrum (MFCC) (step 5 above)

### Device 1: 192.168.0.109

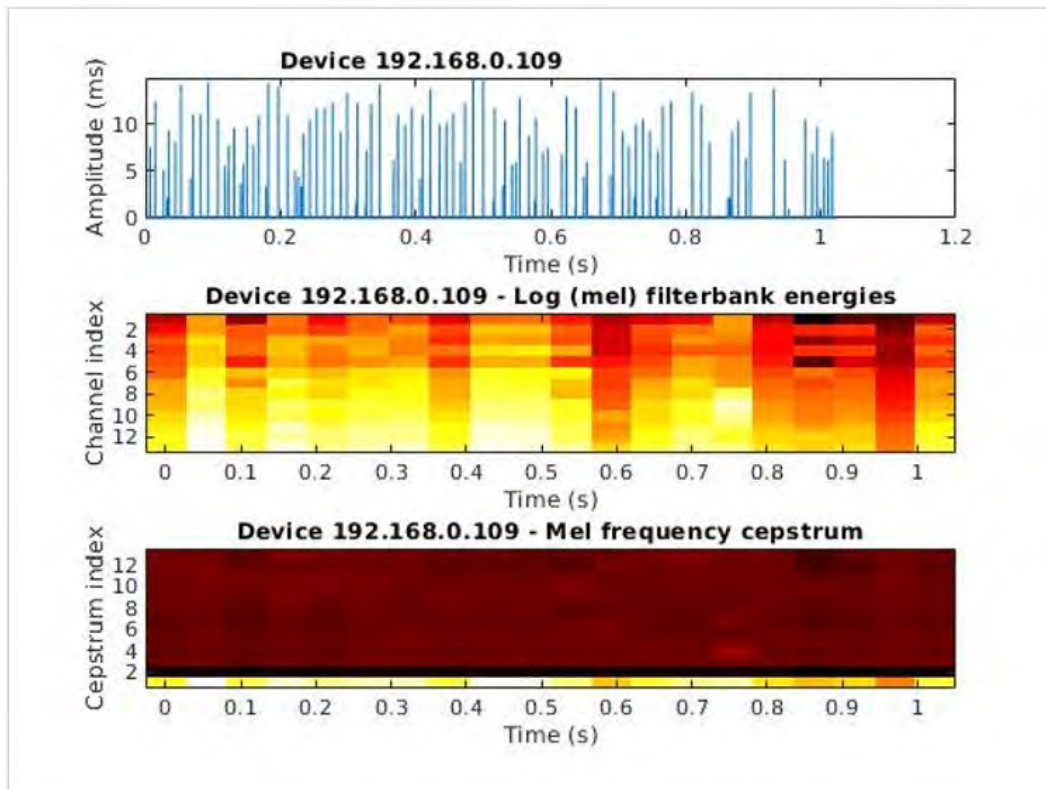


Figure 7.6: Device 1 Time and MelPlots.

From the plots, one can interpret that the Filter bank energy plots (Log of the Mel Coefficients) are different for each of the devices. Since visually, each one is different, we can conclude that Inter-Arrival time is a possible feature to uniquely identify the devices.

Also, it has been decided that the DCT coefficients will be disregarded, and the logs output will be used as the input to the CNN. This is because taking the DCT has removed important information that can be used to distinguish the devices, since its plot across the devices look similar.

### 7.3. CNN Architecture and Results

In this section we present the CNN Model architecture, and the results of training and running the model.

As discussed in section 7.2, the input to the CNN are the Log Mel Specs values. The selected window size is 1 second. That is, the model is trained by input size of 1 second each. The result of the Mel Specs is (13 by 20) for each second. Therefore, this means

that the input to the CNN is of size 13 by 20. The input has been normalized before providing it to the model. The CNN Model and its layers is described in Figure 7.7.

Layer (type)	Output Shape	Param #
conv2d_27 (Conv2D)	(None, 13, 20, 16)	160
conv2d_28 (Conv2D)	(None, 13, 20, 32)	4640
conv2d_29 (Conv2D)	(None, 13, 20, 64)	18496
max_pooling2d_9 (MaxPooling2D)	(None, 6, 10, 64)	0
dropout_9 (Dropout)	(None, 6, 10, 64)	0
flatten_9 (Flatten)	(None, 3840)	0
dense_18 (Dense)	(None, 16)	61456
dense_19 (Dense)	(None, 18)	306

Figure 7.7: CNN Model Before Hypertuning.

The Architecture of the CNN model is presented in Figure 7.8.

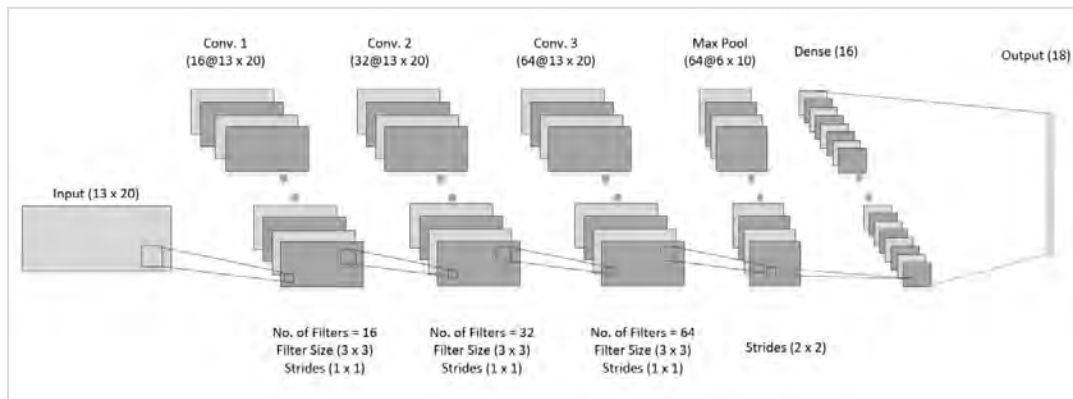


Figure 7.8: CNN Architecture Before Hypertuning.

The model consists of 3 Convolutional layers, followed by a Max Pooling layer, a Flatten layer, and 2 consecutive Dense layers. The output of the model consists of 18 possible outcomes corresponding to the 18 devices.

- The first Convolutional layer consist of 16 filters, each of size (3,3). The activation function applied is *relu*. The stride is set to (1,1) and is padded so as to make the size of the input same as the size of the output. Therefore, the output of the first layer is (13, 20, 64). This results in 16 Feature Maps,

each with size 13 x 20. Therefore, the number of trainable parameters is 160:  $160 = 16 \text{ (bias)} + 3 * 3 \text{ (kernel)} * 16 \text{ (filters)}$

- The second Convolutional layer consist of 32 filters, each of size (3,3). The activation function applied is *relu*. The stride is set to (1,1) and is padded so as to make the size of the input same as the size of the output. Therefore, the output of the first layer is (13, 20, 64). This results in 16 Feature Maps, each with size 13 x 20. Therefore, the number of trainable parameters is 4,640:  $4,640 = 32 \text{ (bias)} + 3 * 3 \text{ (kernel)} * 32 \text{ (filters)} * 16 \text{ (Feature Maps)}$
- The third Convolutional layer consist of 64 filters, each of size (3,3). The activation function applied is *relu*. The stride is set to (1,1) and is padded so as to make the size of the input same as the size of the output. Therefore, the output of the first layer is (13, 20, 64). This results in 64 Feature Maps, each with size 13 x 20. Therefore, the number of trainable parameters is 18,496:  $18,496 = 64 \text{ (bias)} + 3 * 3 \text{ (kernel)} * 64 \text{ (filters)} * 32 \text{ (Feature Maps)}$
- The Max Pooling 2D layer will decrease the dimensions of the data by taking the average of stride set to (2,2).
- The flatten layer is to convert the data into a vector and prepare it for the following Dense layer.
- The first Dense layer consist of 16 units, and the activation function applied is *relu*.
- The second Dense layer consist of 18 units, and the activation function applied is *Softmax* (Classification Layer).

The training parameters of the model were as follows. The model was trained using 30 epochs. This is because the model continued showing decreased validation loss until the number of epochs were around 30. The learning rate set for the model was using the default learning rate in the keras classifier, which was 0.01. The optimizer used for the training the model was *adam* optimizer and was chosen since the work performed in this area showed that this optimizer is commonly used. Finally, the model was trained with a training and validation split of 70%, 30% respectively.

One can interpret from the results that as the number of epochs increased, both the accuracy and the validation accuracy of the model also increased. Also, the

validation loss was also decreasing indicating that the model is not over-fitted. Figure 7.9 that as the number of epochs increased, both the accuracy and the validation accuracy of the model also increased. Also, the validation loss was following the same shape as the training loss indicating that the model is not over-fitted.

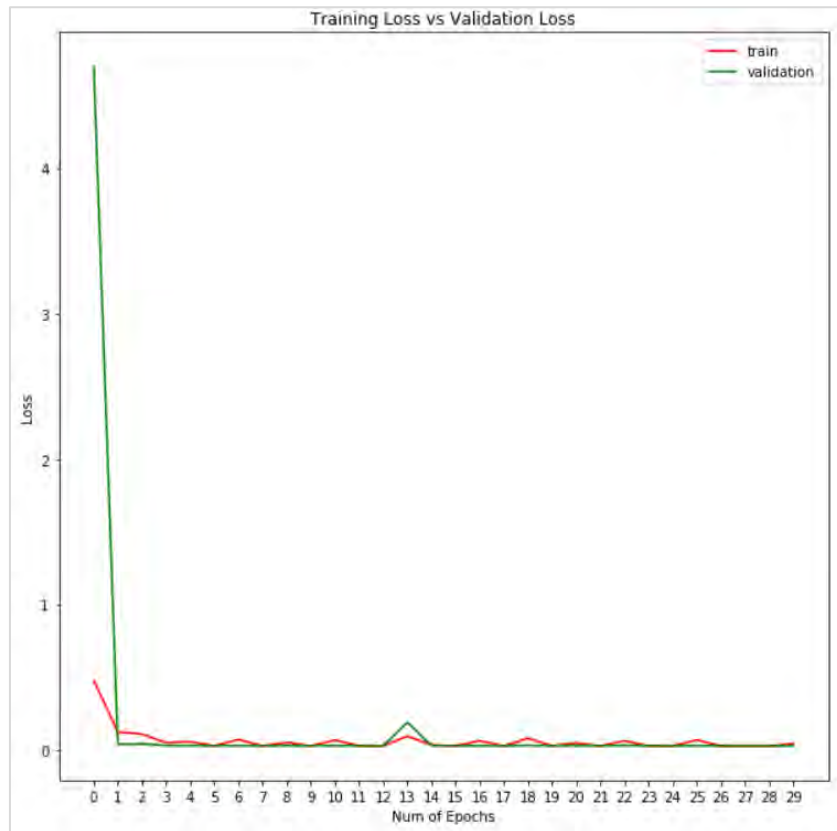


Figure 7.9: Training vs Validation Loss Before Hypertuning.

After training, the model has been used to predict new set (testing data). The results of this are shown in Table 7.1 showing the goodness of measure results. As shown in the table, 13 of the devices showed an F1-score of above 0.99 and 5 between 0.95 and 0.97, indicating that this model is an accurate model for classifying between the devices.

Table 7.1: CNN Before Hypertuning Classification Report.

Device	Precision	Recall	F1-Score
0	1	1	1
1	1	1	1
2	1	1	1
3	1	1	1
4	0.99	1	0.99
5	1	0.98	0.99
6	0.99	0.99	0.99
7	0.99	0.95	0.97
8	0.95	1	0.97
9	1	0.99	0.99
10	0.99	0.99	0.99
11	0.99	1	0.99
12	1	0.97	0.99
13	0.93	1	0.96
14	1	0.91	0.95
15	0.94	0.99	0.97
16	1	0.92	0.96
17	0.93	1	0.97
<b>Micro Average</b>	0.98	0.98	0.98
<b>Macro Average</b>	0.98	0.98	0.98

The confusion matrix in Figure 7.10 shows the correctly and incorrectly classified devices that the was predicted.

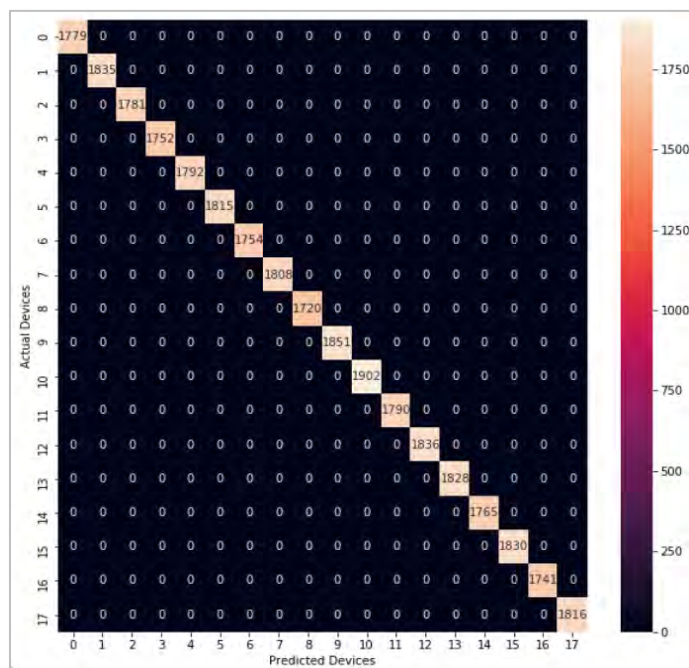


Figure 7.10: Confusion Matrix for Model Before Hypertuning.

In Figure 7.11, one can see that for all devices the ratio of True Positive Rate to False Positive Rate is very high, indicating that the model has correctly identified between the devices. A clearer interpretation of this graph is shown in

Table 7.2 that present the AUC for each of the device's plot.

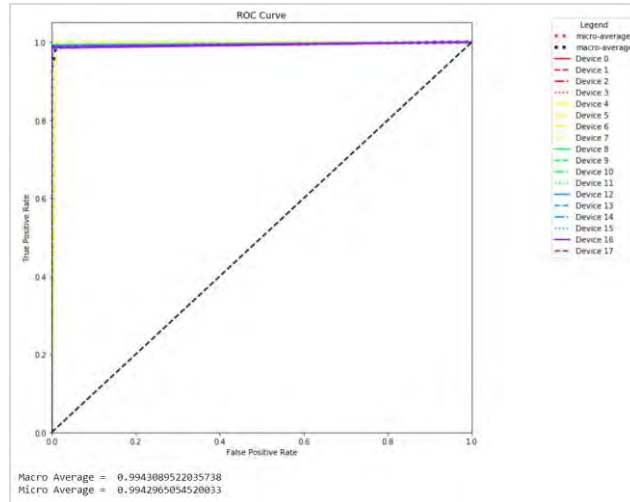


Figure 7.11: ROC Curve Before Hypertuning.

Table 7.2: CNN Model Before Hypertuning AUC Values.

Device Number	AUC value
Device 0	0.9927
Device 1	0.9946
Device 2	0.9935
Device 3	0.9954
Device 4	0.9947
Device 5	0.9945
Device 6	0.9943
Device 7	0.9942
Device 8	0.9965
Device 9	0.9949
Device 10	0.9934
Device 11	0.9939
Device 12	0.9948
Device 13	0.9932
Device 14	0.9952
Device 15	0.9951
Device 16	0.9922
Device 17	0.9945
Macro Average	0.9943
Micro Average	0.9942

However, aiming to achieve more accurate results, hypertuning was performed to get the best parameters in the model. In the next section, we present the hypertuning process and the mode results after considering it.

#### 7.4. Hypertuning

Hypertuning in machine learning is the process of tuning the model hyperparameters in order to select the best optimal values for the learning process. The hyperparameters that can be tuned in a CNN model are the number of filters, filter size, learning rate, number of units in dense layers, activation functions etc. Typically, in the hypertuning processes, range of parameters are given, and the tuning process will provide the combination of them for the model that gives the best results [107].

##### Hypertuning Parameters:

The tuning method used is Hyperband tuner [108]. Hyperband tuning is one of the tuning techniques that will train multiple models for small number of epochs and then selects the best model and resume its training for a greater number of epochs. best selected by the tuner is presented in Table 7.3.

Table 7.3: Hypertuning Results.

CNN Layer	Tuning Parameters	Range of Options	Best Results
<b>Conv. Layer 1</b>	Number of filters	[16, 32, 64]	64
	Kernel size	[1, 3]	(1, 1)
<b>Conv. Layer 2</b>	Number of filters	[16, 32, 64]	64
	Kernel size	[1, 3]	(3, 3)
<b>Conv. Layer 3</b>	Number of filters	[16, 32, 64]	64
	Kernel size	[1, 3]	(1, 1)
<b>Dropout layer</b>	Dropout value	min = 0.0 max = 0.2 step = 0.1	0.0
<b>Dense Layer 1</b>	Number of units	min = 32 max = 256 step = 32	64
	Activation Function	[relu, sigmoid]	relu
<b>Hyperparametrs</b>	Learning Rate	min = 1 e-4 max = 1 e-2 Sampling = Log	0.00309826

After tuning, the optimum parameters were picked, and the model was reconstructed using the new values. The model was trained again and used to perform the predict of the test set again. In Figure 7.12, the model after tuning is shown. The architecture of the tuned model is presented in Figure 7.13.

Model: "sequential"		
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 13, 20, 64)	128
conv2d_1 (Conv2D)	(None, 13, 20, 64)	36928
conv2d_2 (Conv2D)	(None, 13, 20, 64)	4160
max_pooling2d (MaxPooling2D)	(None, 6, 10, 64)	0
dropout (Dropout)	(None, 6, 10, 64)	0
flatten (Flatten)	(None, 3840)	0
dense (Dense)	(None, 64)	245824
dense_1 (Dense)	(None, 18)	1170

Figure 7.12: CNN Model After Hypertuning.

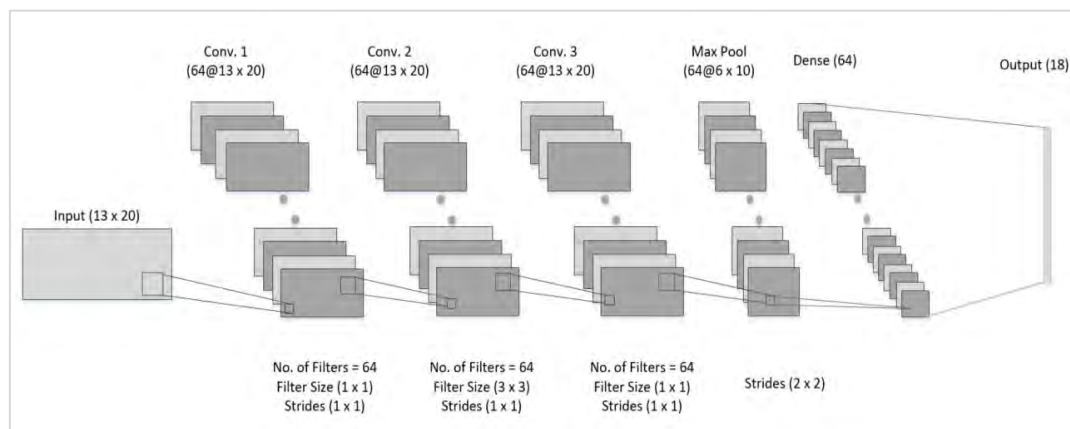


Figure 7.13: CNN Model Architecture After Hypertuning.

The model consists of 3 Convolutional layers, followed by a Max Pooling layer, a Flatten layer, and 2 consecutive Dense layers. The output of the model consists of 18 possible outcomes corresponding to the 18 devices.

- The first Convolutional layer consist of 64 filters, each of size (1,1). The activation function applied is *relu*. The stride is set to (1,1) and is padded so as to make the size of the input same as the size of the output. Therefore, the output of the first layer is (13, 20, 64). This results in 16 Feature Maps, each with size 13 x 20.  
Therefore, the number of trainable parameters is 128:  $128 = 64 \text{ (bias)} + 1 * 1 \text{ (kernel)} * 64 \text{ (filters)}$
- The second Convolutional layer consist of 64 filters, each of size (3,3). The activation function applied is *relu*. The stride is set to (1,1) and is padded so as to make the size of the input same as the size of the output. Therefore, the output of the first layer is (13, 20, 64). This results in 16 Feature Maps, each with size 13 x 20.  
Therefore, the number of trainable parameters is 36,928:  $36,928 = 64 \text{ (bias)} + 3*3 \text{ (kernel)} * 64 \text{ (filters)} * 64 \text{ (Feature Maps)}$
- The third Convolutional layer consist of 64 filters, each of size 1 by 1. The activation function applied is *relu*. The stride is set to (1,1) and is padded so as to make the size of the input same as the size of the output. Therefore, the output of the first layer is (13, 20, 64). Therefore, the output of the first layer is (13, 20, 64). This results in 64 Feature Maps, each with size 13 x 20.  
Therefore, the number of trainable parameters is 4,160:  $4,160 = 64 \text{ (bias)} + 1*1 \text{ (kernel)} * 64 \text{ (filters)} * 64 \text{ (Feature Maps)}$
- The Max Pooling layer will decrease the dimensions of the data by taking the average of stride set to (2,2).
- The flatten layer is to convert the data into a vector and prepare it for the following Dense layer.
- The first Dense layer consist of 64 units, and the activation function applied is *relu*.
- The second Dense layer consist of 18 units, and the activation function applied is *Softmax* (Classification Layer).

**7.4.2 Hypertuning model results.** The model has then been compiled using *adam* optimizer and trained for 30 epochs. The model has been split in 70% 30%

training and validation split, respectively. The training batch size was set to 32. The plot of training loss versus validation loss is shown in Figure 7.14. It shows that as the number of epochs increased, both the accuracy and the validation accuracy of the model also increased. Also, the validation loss was somehow following the same shape as the training loss (except for a spike at one of the epochs) indicating that the model is not over-fitted.

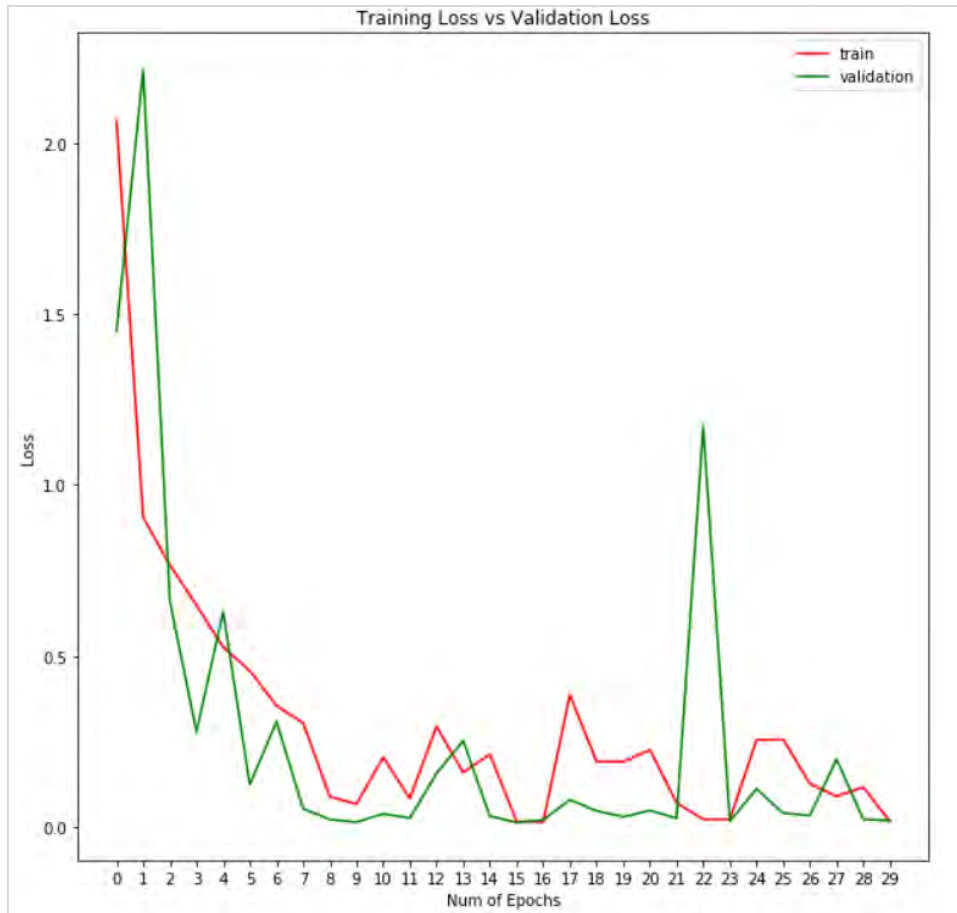


Figure 7.14: Training versus Validation Loss After Hypertuning.

**Classification Report:**

- After training, the model has been used to predict new set (testing data). The results of this are shown in Table 7.4 showing the goodness of measure results. As shown in the table, 16 of the devices showed an F1-score of above 0.99 and 2 around 0.99, indicating that this model is an accurate model for classifying between the devices, and the hypertuned parameters provided better results.

Table 7.4: CNN After Hypertuning Classification Report.

Device	Precision	Recall	F1-Score
0	1	0.99	0.99
1	0.99	1	0.99
2	1	1	1
3	1	1	1
4	1	1	1
5	1	1	1
6	1	1	1
7	1	1	1
8	1	1	1
9	1	1	1
10	1	1	1
11	1	1	1
12	1	1	1
13	1	1	1
14	1	1	1
15	1	1	1
16	1	1	1
17	1	1	1
<b>Micro Average</b>	1	1	1
<b>Macro Average</b>	1	1	1

**ROC Curve, K-fold, Confusion Matrix:**

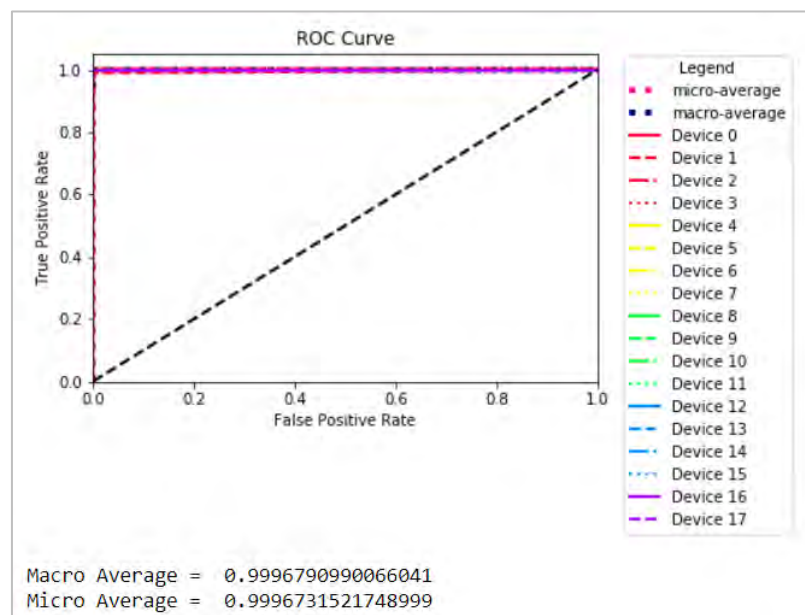


Figure 7.15: CNN Model ROC Curve.

In Figure 7.15, one can see that for all devices the ratio of True Positive Rate to False Positive Rate is very high, indicating that the model has correctly identified between the devices. A clearer interpretation of this graph is shown in

Table 7.5 that present the AUC for each of the device's plot. As shown in the table, most of the devices have an AUC of 1.0, indicating a highly accurate model.

Table 7.5: CNN Model AUC Values.

Device Number	AUC value
Device 0	0.9996
Device 1	0.9945
Device 2	1.0
Device 3	1.0
Device 4	1.0
Device 5	1.0
Device 6	1.0
Device 7	1.0
Device 8	1.0
Device 9	1.0
Device 10	1.0
Device 11	1.0
Device 12	1.0
Device 13	1.0
Device 14	1.0
Device 15	1.0
Device 16	1.0
Device 17	1.0
Macro Average	0.9996
Micro Average	0.9996

In order to test if this model has been over-fitted, K-Fold has been implemented. The confusion matrix associated with each of the 10 fold run is done. The best accuracy is shown in Figure 7.16. The rest of the folds confusion matrices is shown in Appendix C.

**K-Fold and Confusion Matrix:**

Fold 3, accuracy: 0.9995

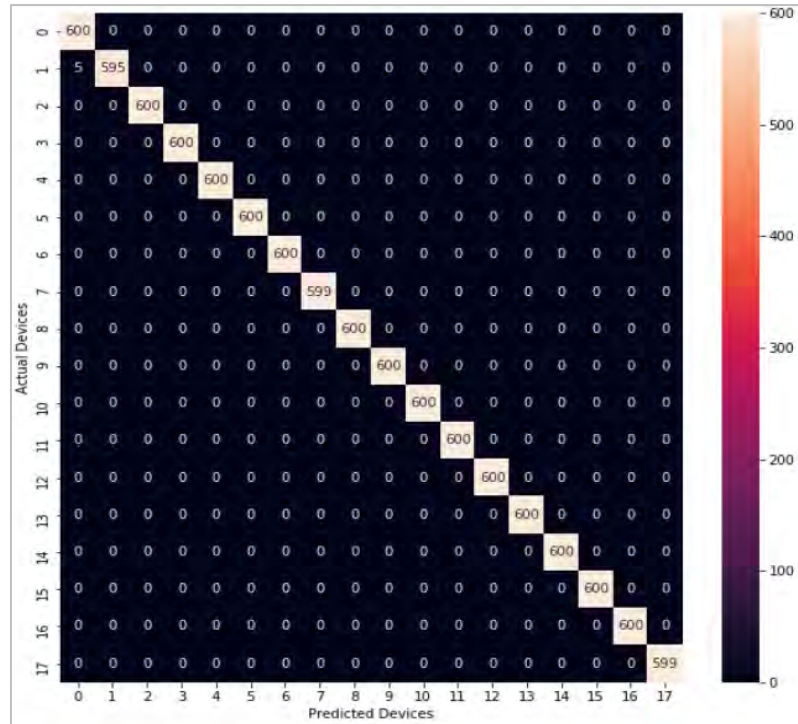


Figure 7.16: Fold 3 Confusion Matrix.

As shown across all the confusion matrices, each of the run showed that almost all devices were correctly identified Only 5 to 7 devices across all were incorrectly identified. Also, from the accuracy of each run, all accuracies were above 0.9993 indicating a very accurate model that is not overfitted. A detailed report of the accuracy, precision, recall, and F1-Score for each fold is shown in Table 7.6. It can be shown that almost all values are more than 0.99 for all the four measures indicating a reliable model.

Table 7.6: K-Fold Results for Model After Hypertuning.

Metric	Accuracy	Precision	Recall	F1-score
Mean	0.9993 (0.0001)	0.9900	0.9971	0.9959

## 7.5. Discussion

In this section, the convolutional aspect of the model is explained in detail. This is in terms of presenting the filters of each of the three convolutional layers, presenting the results of providing an input to these filters, and how they are used to distinguish between the devices. There are several plots that will fall under this section, and they are placed in Appendix A.

**7.5.1. Convolutional layer 1.** Recalling from the previous section, the first convolutional layer shown in Figure 7.17 consist of:

- 64 filters,
- filter size of (1,1),
- stride size of (1,1),
- padded such that the input and output have same size.

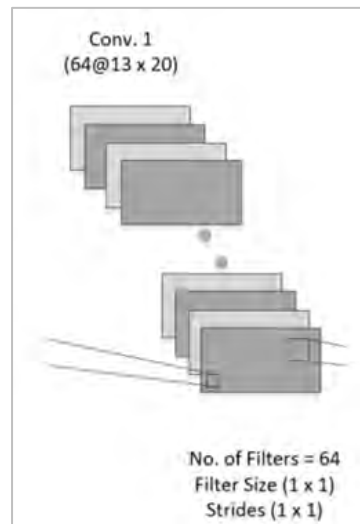


Figure 7.17: Convolutional Layer 1.

Since the first layer consist of 64 filters each if size (1,1), the result of the first layer is 64 Feature Maps. Since the setting is padded, the output will consist of (13, 20) 64 times (since strides is set to 1).

Since the filters in the first layer is of size (1,1), and is only a single channel, that is, the values are only between 0 and 1, every filter application involves multiplying the pixel value in the input by a value between 0 to 1. This means that each input is converted to another scaler value that is either 0 (black), 1 (white), or a value in between (shades of gray). And since there are 64 of these filters in the first layer, the model is attempting various possible graying of each input, i.e, producing different feature maps. Therefore, the filters at layer are attempting to pick up different features of the input.

To look at the distribution of these filters and to see what each of them is doing, a histogram has been plotted for each of the filters in layer 1 shown in Figure 7.18. As shown from the plot, around 9 of the 64 filters are centered around value 0.2 and 0.3, 8 of the 64 filters around value 0.6, and 5 filters around value 0 (darkens the input to

black), and 5 filters around value 1 (lightens the input to white), and the rest are in between.

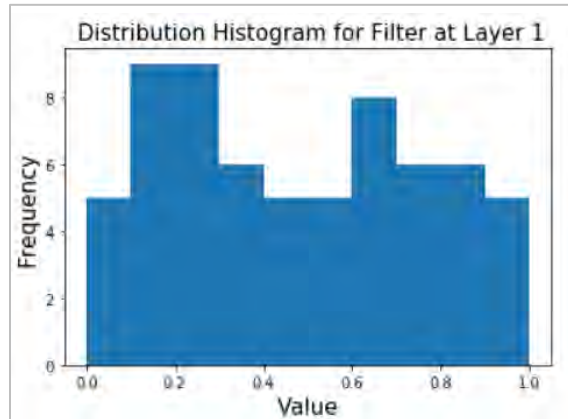


Figure 7.18: Filters Distribution for Convolutional Layer 1.

**7.5.2. Convolutional layer 2.** Recalling from the previous section, the first convolutional layer shown in Figure 7.19 consist of:

- 64 filters,
- filter size of (3,3),
- stride size of (1,1),
- padded such that the input and output have same size.

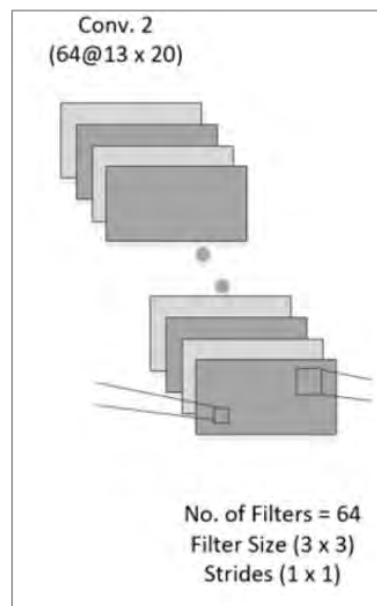


Figure 7.19: Convolutional Layer 2.

Each of the filters consist of (3,3) matrix, i.e., 9 values. They are plotted and shown in Figure 7.20 and Figure 7.21.

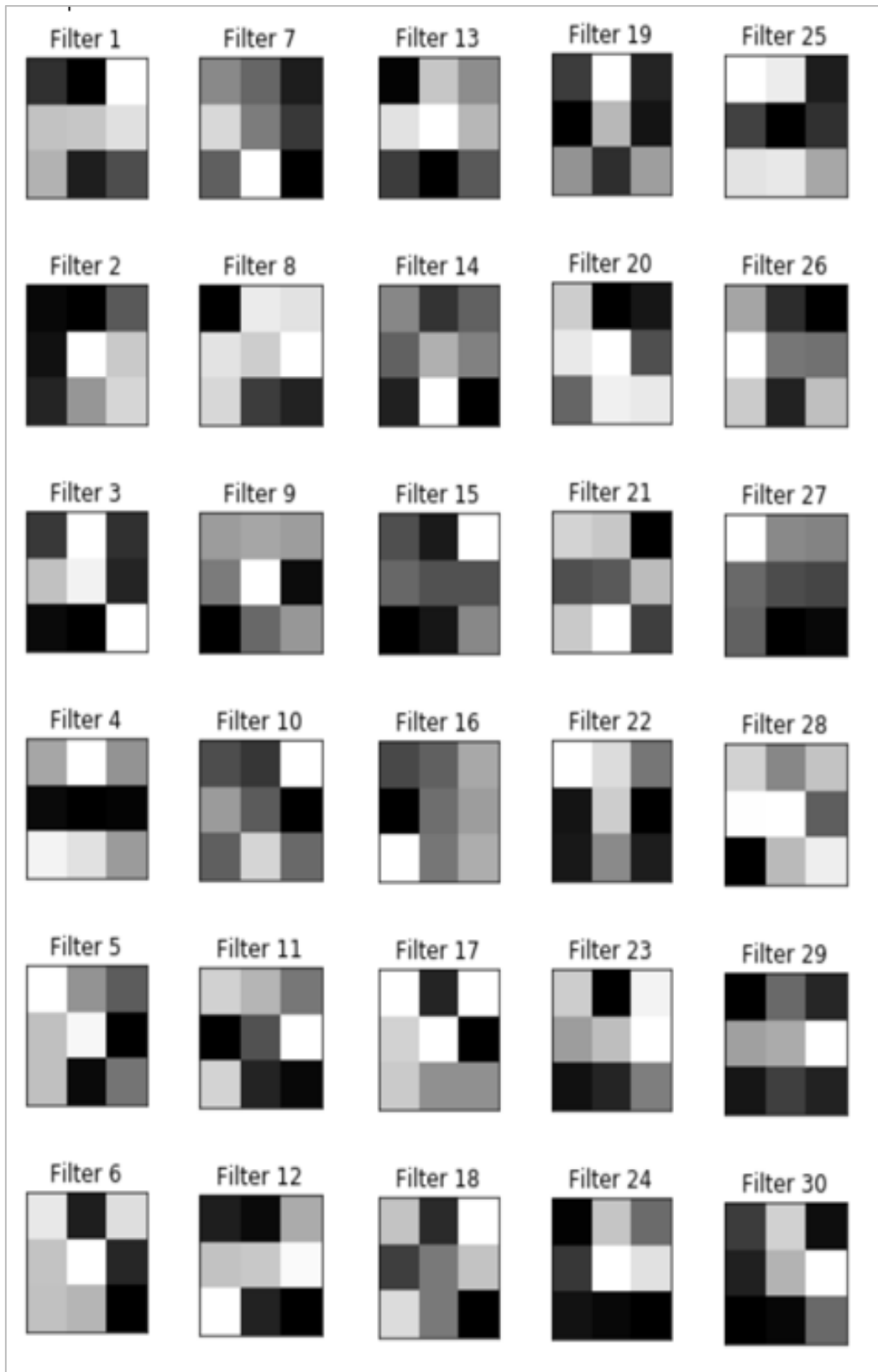


Figure 7.20: Layer2 Filters 1 – 30.

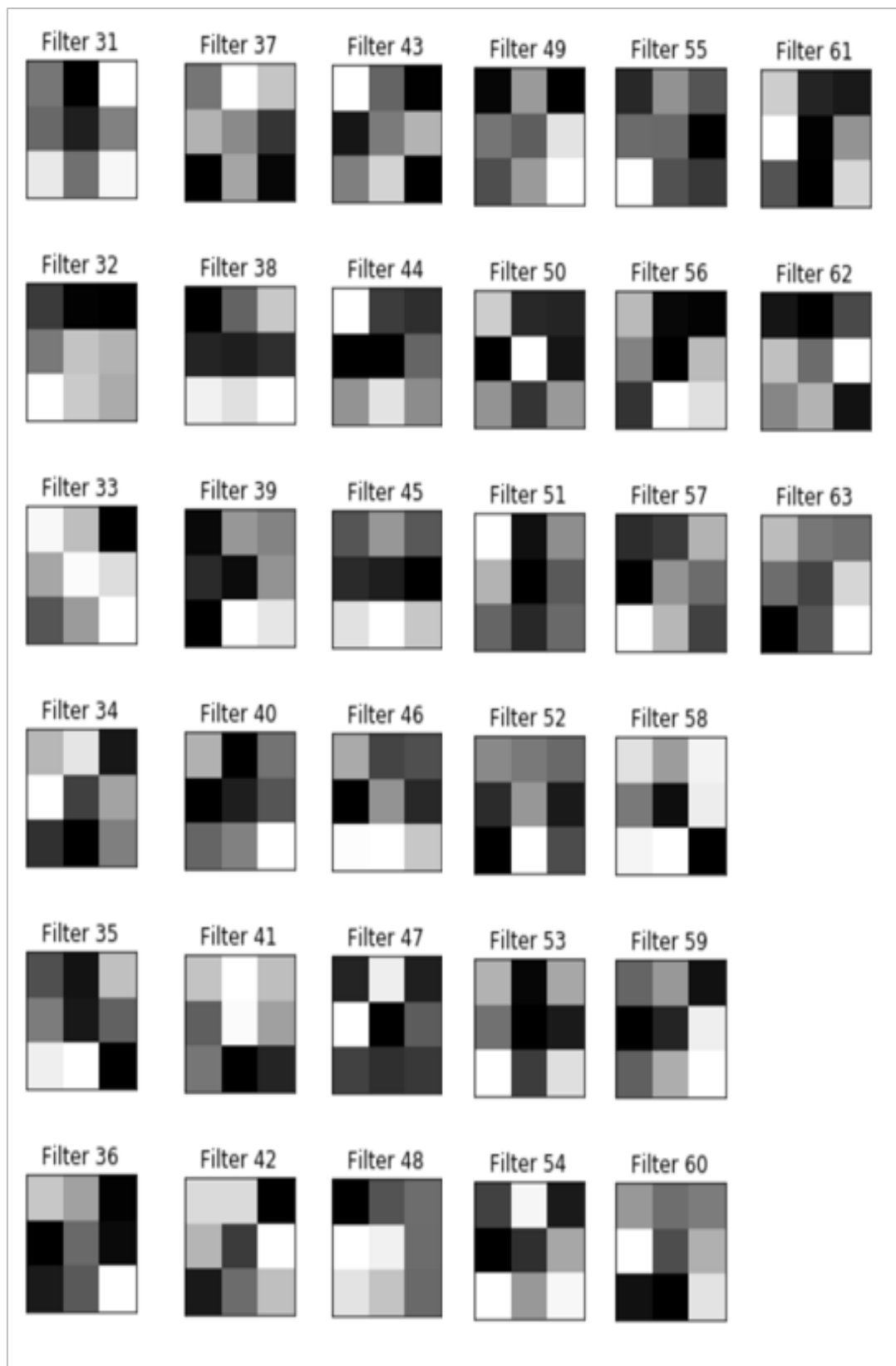


Figure 7.21: Layer 2 Filters 31 – 63.

Since it is difficult to look at similarities across 64 filters, we perform K-Means clustering to them to group filters that are similar together. In this case, we can visually interpret what does filters look like, how similar they are and what they are doing.

**K-Means Clustering:**

The filter values have been extracted from the trained model and gathered to perform k-means clustering algorithm [109]. In order to get the optimum number of clusters, the elbow method has been performed. The results of this method showed that the optimum number of clusters is 9 as shown in Figure 7.22.

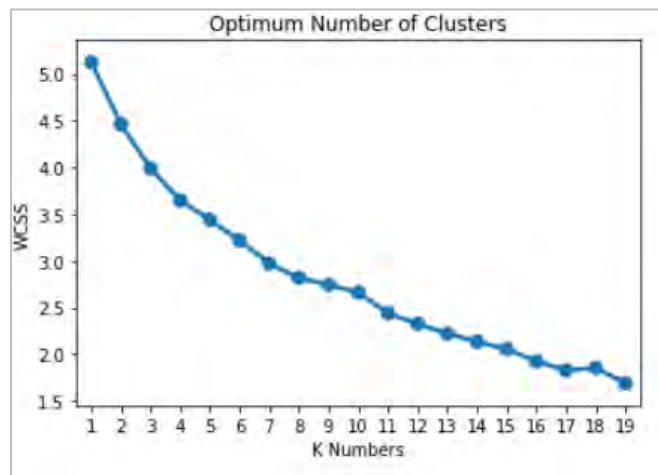


Figure 7.22: Optimum Number of K-Means Clusters.

After fitting the model, it has placed the 64 filters in 9 clusters. The result of this clustering is shown in Table 7.7.

Table 7.7: K-Mean Clustering Result.

Cluster No.	Filter Number
1	[29, 40, 46, 52, 54, 57]
2	[ 5, 31, 35, 36, 56]
3	[ 0, 1, 7, 8, 11, 14, 24, 44, 63]
4	[ 3, 9, 12, 23, 27, 30, 32, 41, 42, 49, 50]
5	[13, 17, 20, 22, 25, 37, 53, 55, 62]
6	[ 2, 10, 15, 19, 21, 33, 51]
7	[ 4, 18, 28, 39, 47, 59]
8	[ 6, 16, 26, 34, 43, 45, 60, 61]
9	[38, 48, 58]

From the above table, we can have a clearer view of the filters that are grouped together and are similar. In order to understand the differences among these 9 groups, we plot each of the clusters. This is done by taking the mean of each of the 9 clusters. These cluster plots are shown in Figure 7.23. By looking at the plots, one can interpret that each cluster is different than the other. This is an indication that the cluster (or the filters within each clusters) are trying to pick up different features. Every cluster have shades of gray image colors of white, black, gray indicating that it each one is trying to (gray out) or trying various possible graying to pick up on different features.

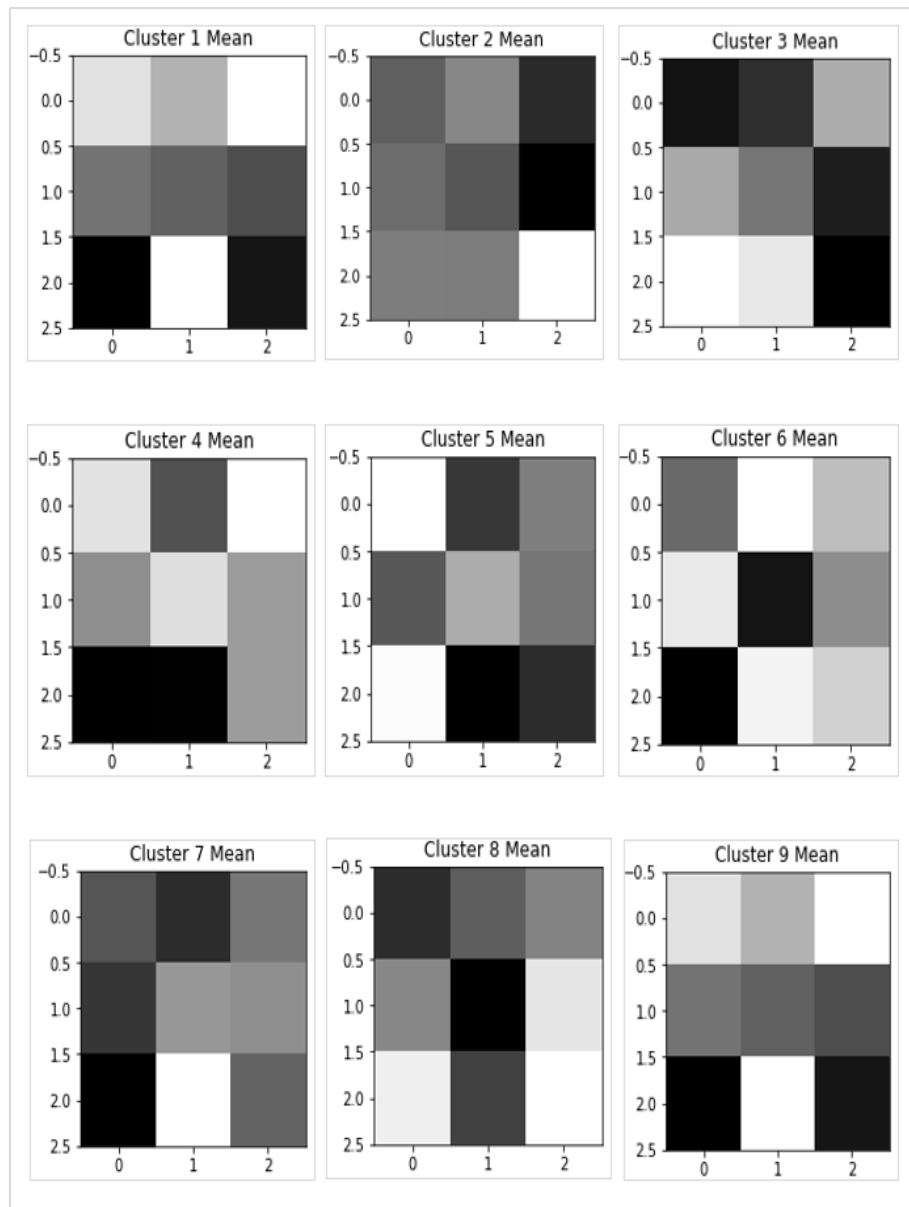


Figure 7.23: K-Means Cluster Means Plot.

In order to visualize what these filters are performing on the input data, the result of applying these filters to a single input from each device is plot. This is shown in Figure 7.24 and Figure 7.25. The input for the rest of the devices is shown in Appendix D. Each of the figures is an 8 by 8 matrix that is corresponding to the 64 sized filter. It can be shown that the filters constructed variations, as some areas are darker than the others. In some areas it is darker than other, aiming to pick up different features in the input.



Figure 7.24: Feature Map of Layer 2 Filter on Device 0.

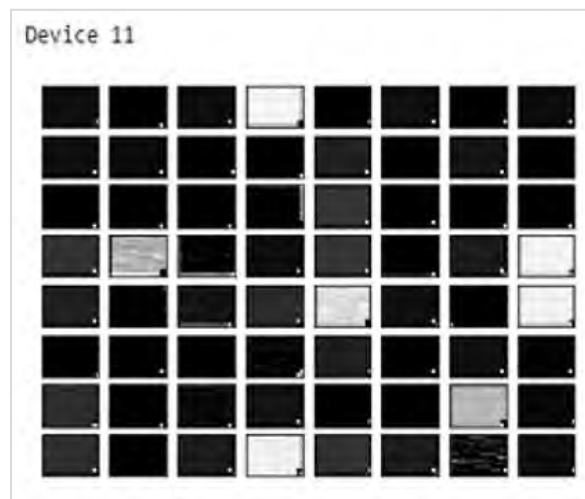


Figure 7.25: Feature Map of Layer 2 Filter on Device 11.

**7.5.3. Convolutional layer 3.** The third convolutional layer shown in Figure 7.26 consist of:

- 64 filters,
- filter size of (3,3),
- stride size of (1,1),
- padded such that the input and output have same size.

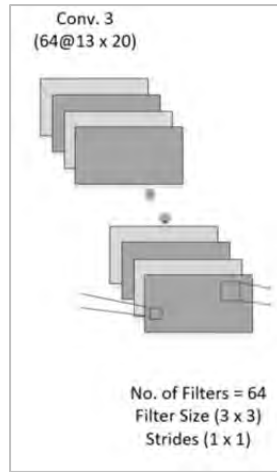


Figure 7.26: Convolutional Layer 3.

The output of this layer will consist of 64 Feature Maps, each with size of 64. To understand each of these maps, their frequency distribution (histogram has been plot) and are shown in Figure 7.27, Figure 7.28, and Figure 7.29.

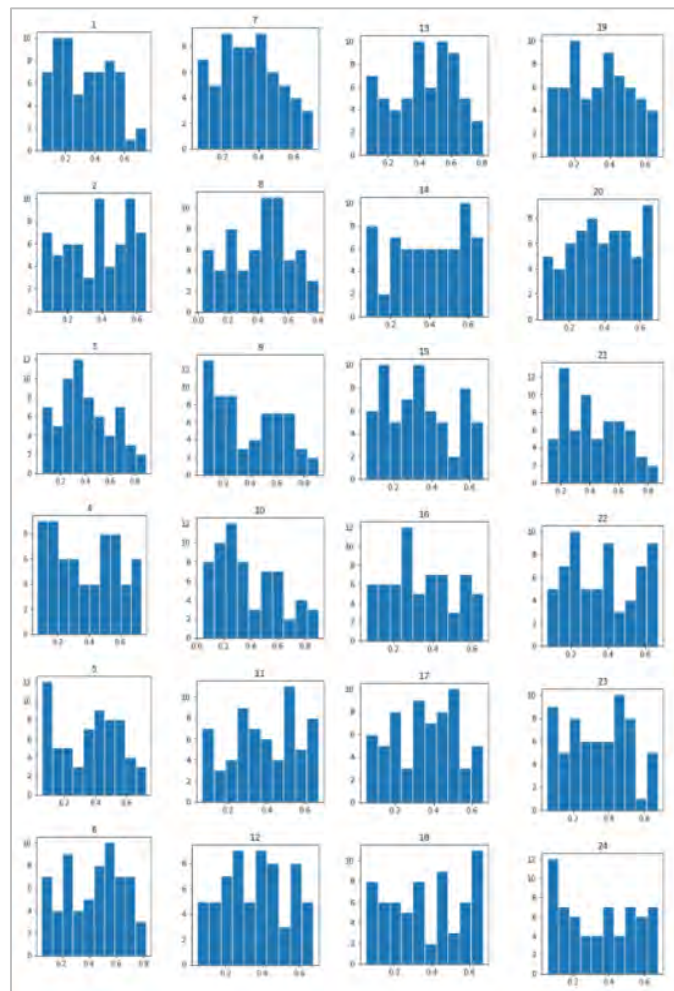


Figure 7.27: Layer 3 Filters 1 - 24 Histogram.

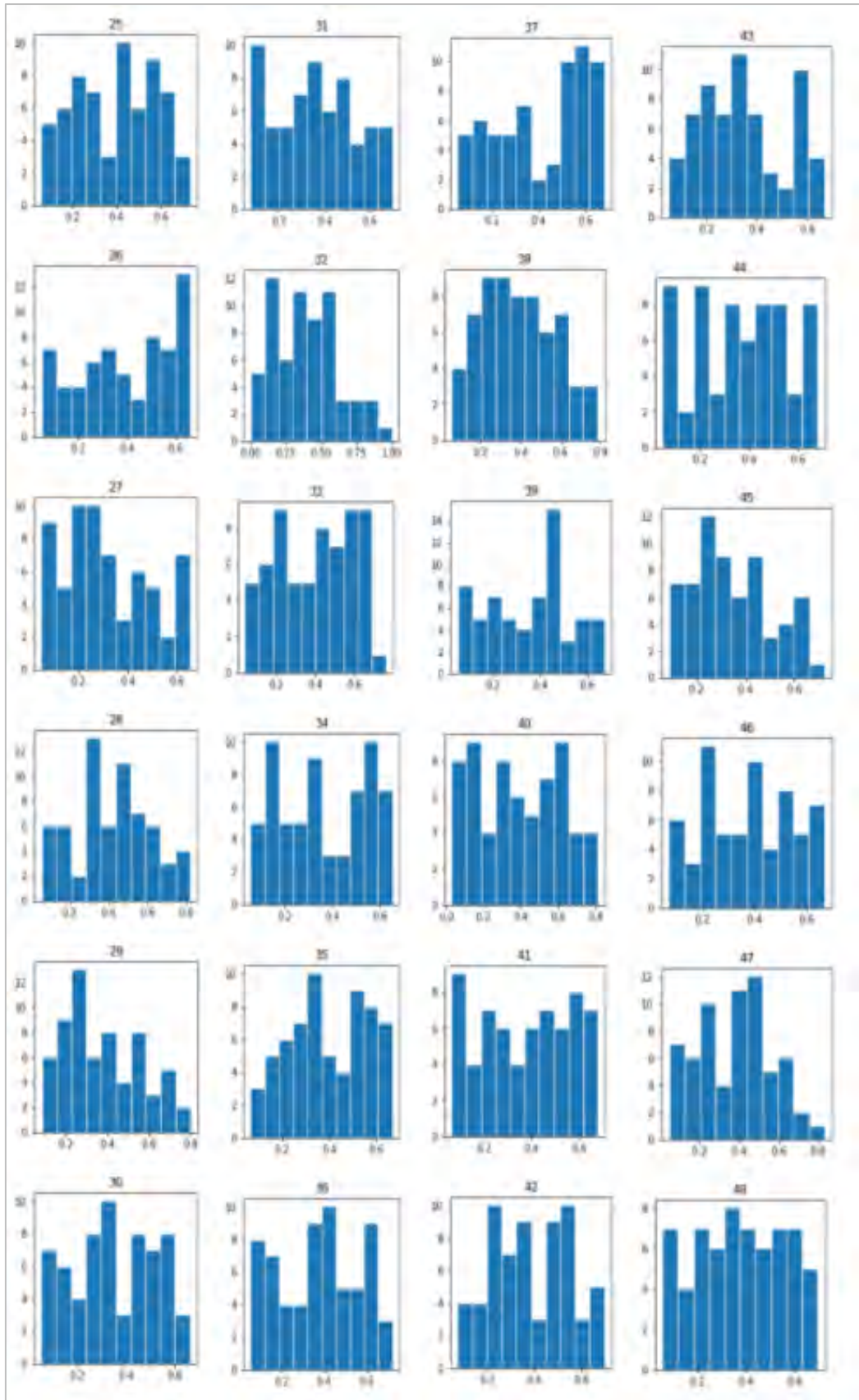


Figure 7.28: Layer 3 Filters 25 - 48 Histogram.

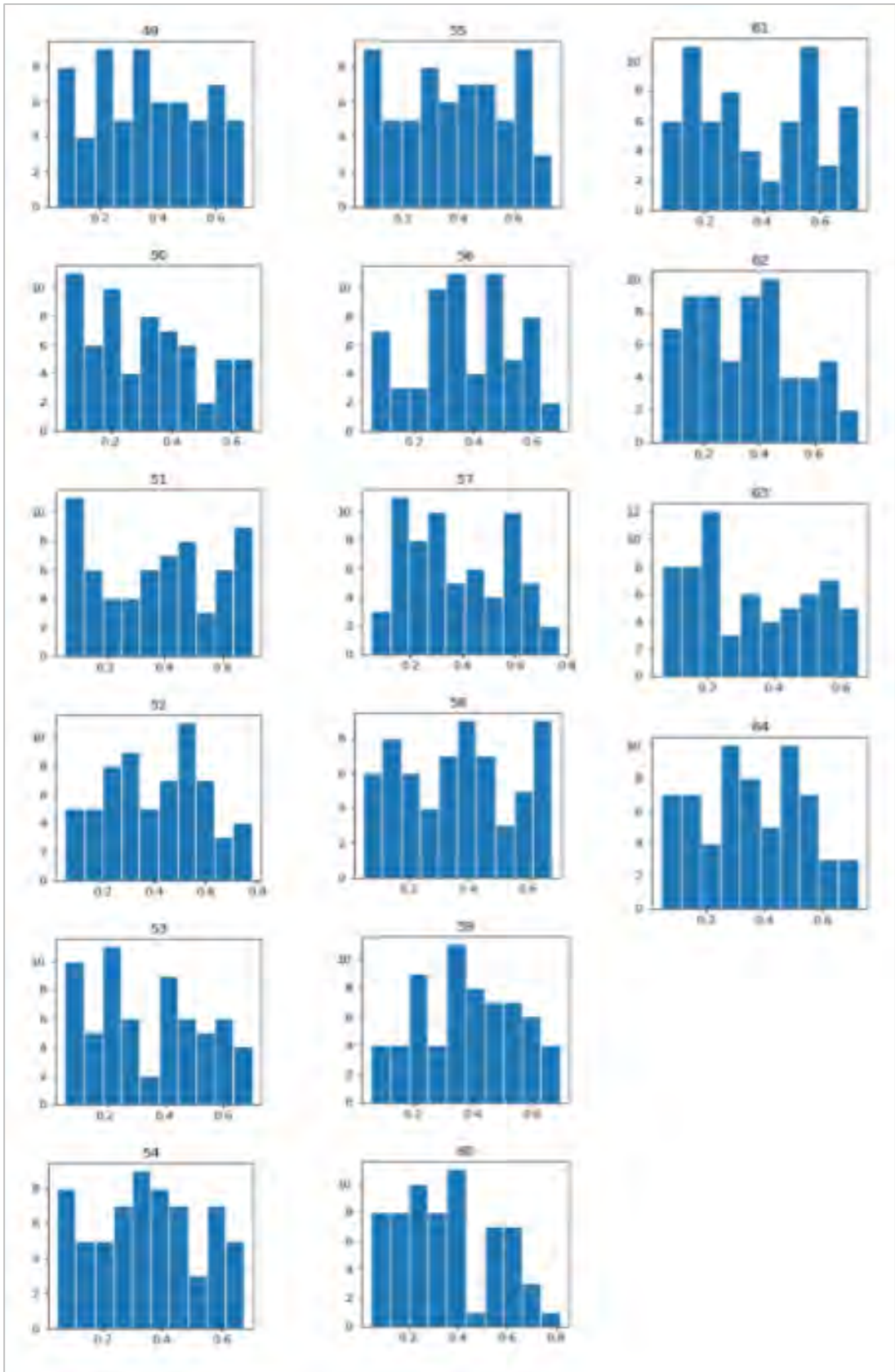


Figure 7.29: Layer 3 Filters 49 - 64 Histogram.

In order to better interpret the 64 filters, the entropy has been calculated and its distribution is plot. The entropy is a measure that will be used to tell us how much information is present. In the context of the problem, the entropy of each filter will tell us how much information is present in each of the 64 filters. In Figure 7.30, the entropy distribution is plot. The entropy is somehow creating an envelope to the shape of the distribution of these filters, in order to give us an idea of what they are doing. From the plot, one can see that the entropy is following a normal distribution shape centred at -0.44. An example of a filter that has an entropy of -0.44 like the one shown in Figure 7.31. These entropy values for each filter are shown in Table 7.8.

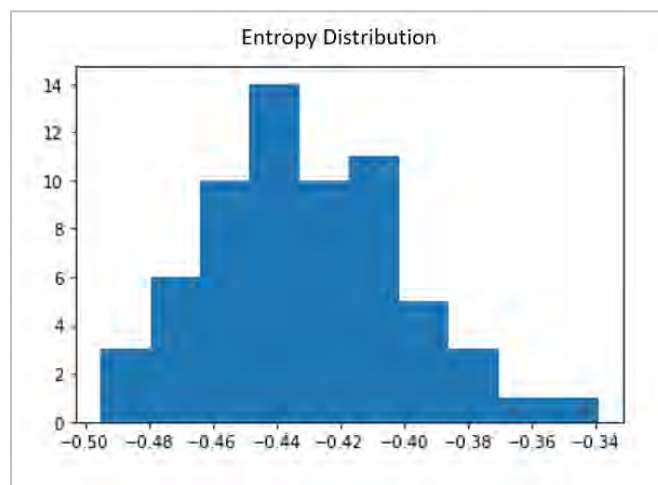


Figure 7.30: Entropy Distribution for Filters in Third Layer.

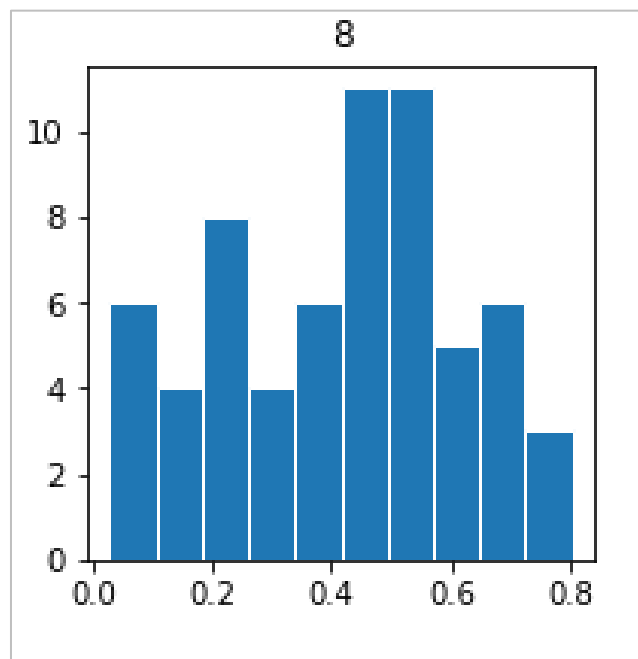


Figure 7.31: Filter 8 Distribution.

Table 7.8: Entropy Values for Filter in Third Layer.

Filter No.	Entropy Value	Filter No.	Entropy Value
1	-0.46055	34	-0.41202
2	-0.40509	35	-0.40415
3	-0.46325	36	-0.44346
4	-0.41043	37	-0.38707
5	-0.45249	38	-0.43991
6	-0.43997	39	-0.45030
7	-0.43646	40	-0.42836
8	-0.44986	41	-0.39455
9	-0.47226	42	-0.43686
10	-0.46213	43	-0.44861
11	-0.39879	44	-0.40297
12	-0.42027	45	-0.47130
13	-0.44354	46	-0.41128
14	-0.40075	47	-0.48462
15	-0.42740	48	-0.40899
16	-0.42506	49	-0.41595
17	-0.42639	50	-0.43272
18	-0.36570	51	-0.38635
19	-0.42499	52	-0.43550
20	-0.37143	53	-0.44258
21	-0.46627	54	-0.41679
22	-0.38450	55	-0.43387
23	-0.42930	56	-0.47273
24	-0.40712	57	-0.46481
25	-0.43999	58	-0.37836
26	-0.33904	59	-0.43548
27	-0.41468	60	-0.48003
28	-0.45722	61	-0.42101
29	-0.46706	62	-0.45503
30	-0.44082	63	-0.42937
31	-0.41948	64	-0.44523
32	-0.49510		
33	-0.45613		

## Chapter 8. Convolutional Neural Network – Experiment 2

J. Kotak et al. [71] used Neural Network to distinguish between IoT devices, however, the input to the model was different. In this thesis's work, we leverage the same input used in the literature work and construct a model.

In this chapter, the work in literature is explained in section 8.1. This involves the describing the input, the model, and the results they produced. In section 8.2, our experiment is presented. This presents the input from the ESP devices and the model, the results of the model fitting along with the goodness of measures of the model.

### 8.1. Input, Model, and Results of CNN from Literature

The work is aimed to identify between 10 IoT different devices, such as Amazon Echo, Samsung SmartCam, motion sensor etc. A model was built each class of devices. Each model consisted of an input layer and an output layer and is shown in Figure 8.1. The model is simply a dense layer architecture model.

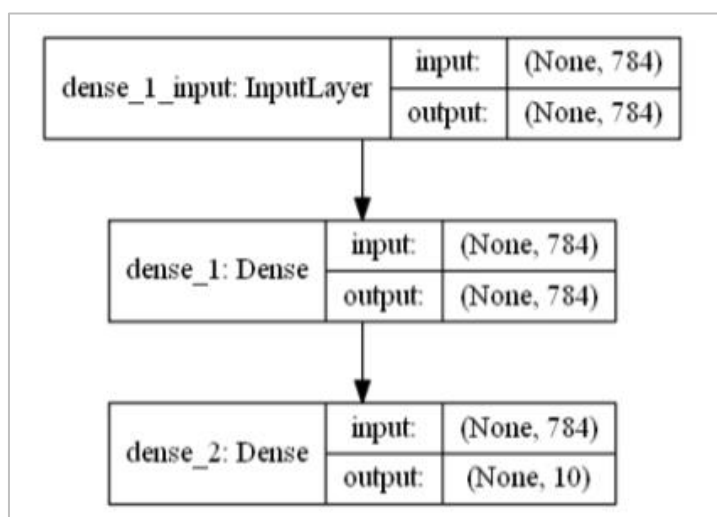


Figure 8.1: Dense Model Architecture [71].

The input to the model has been pre-processed in the following manner:

1. The collected data from the IoT devices were grouped based on the TCP packets that have the same source and destination IP addresses, same source and destination port number, and the same protocol. This will result into multiple pcap files.
2. Then, the pcap files were segregated into files that based on the same MAC address.
3. The files are then converted to binary format.

4. The file size is adjusted so that every binary file has a size of 784 bytes.
5. Then, each binary file is converted to 28 x 28 image (gray scale). This is the input to the model.

The 10 IoT devices are all of different types. The total data collected is 107,808 TCP sessions and the model was trained to 80% training set, 10% testing set, and 10% validation set. The results of the model showed an accuracy of 0.9986 and the confusion matrix for each of the devices is shown in Figure 8.2. As one can see, the model seemed to perform well for detecting the devices.

Actual IoT device/classified as	0	1	2	3	4	5	6	7	8	9
0- Non-IoT devices	2226	0	0	0	0	0	0	0	0	0
1- Amazon Echo	1	812	0	0	0	0	0	0	0	0
2- Samsung SmartCam	2	0	321	0	0	0	0	0	0	0
3- Belkin Wemo switch	0	0	0	365	0	0	0	0	0	0
4- Netatmo Welcome	0	0	0	0	306	1	0	0	0	0
5- Insteon camera	0	0	0	0	0	210	0	0	0	0
6- Withings Aura smart sleep sensor	1	0	0	0	0	0	241	0	0	0
7- Netatmo weather station	0	0	0	0	0	0	0	100	0	0
8- PIX-STAR photoframe	0	0	0	0	0	0	0	0	628	5
9- Belkin Wemo motion sensor	1	0	0	0	0	0	0	0	1	3465

Figure 8.2: Confusion Matrix [71].

## 8.2 Input to the Model and Model Fitting

In this section, we present how the input from the ESP32 devices were processed and how the model was fitted in order to compare with the work done in this chapter.

The following processing to the input was performed:

1. From the captured data files (pcaps) of the 18 ESP32 devices, the files were grouped based on their Source and Destination IP address, resulting in each pcap file containing the sessions from one unique device only.
2. One second data from each of the files were captured. This was done by looking at the time of between the packets. For each approximate of one second, the associated packets in that frame are separated.
3. Since each one second is approximately 1000 bytes (some are less, some are more), and in order to be able to convert it to 28 by 28 input, each one second bytes of data were converted to 784 bytes. This was done by either trimming the bytes or padding them with 0 till they reach 784 bytes.

4. Each of these 784 bytes were converted to binary data.
5. This was performed and gathered for each of the 18 devices, and the 28 by 28 bytes were used as an input to the model. The model constructed is described.

The model architecture followed is 1 model for all devices, multi-class classification that will classify one device versus all others. The architecture of the model is shown in Figure 8.3.

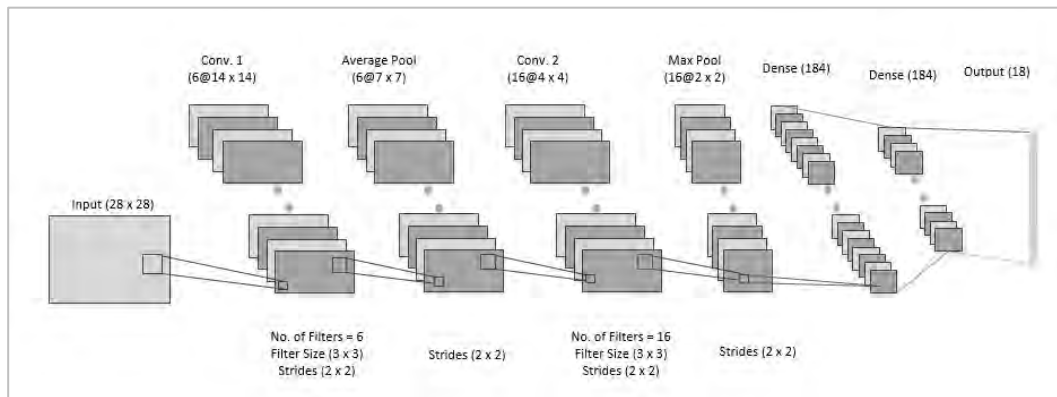


Figure 8.3: Model Architecture [74].

The model was fitted using 500 Epochs, with training set and testing set of split 70%, 30% respectively. The training loss versus validation loss plot is shown in Figure 8.4. It can be seen that validation loss is decreasing as the number of epochs is increasing indicating that the model is not overfitted [110]. The model has been trained till epoch value of 500.

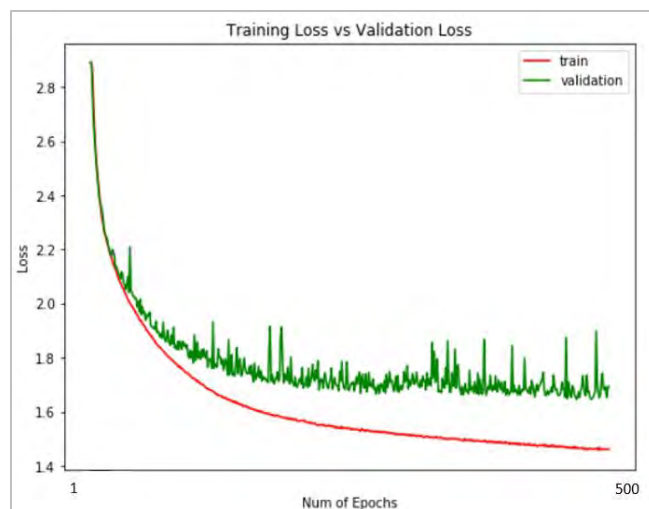


Figure 8.4: Training Loss vs Validation Loss Curve (Binary Input).

The classification report that shows the Precision, Recall, and F1 -Score of the model is shown in Table 8.1. The micro average score is 0.42.

Table 8.1: Classification Report for CNN (Binary Input).

Device	Precision	Recall	F1-Score
0	0.29	0.18	0.23
1	0.22	0.15	0.18
2	0.22	0.08	0.12
3	0.65	0.63	0.64
4	0.42	0.61	0.49
5	0.43	0.44	0.44
6	0.34	0.18	0.24
7	0.75	0.74	0.75
8	0.37	0.36	0.37
9	0.62	0.59	0.6
10	0.88	0.85	0.87
11	0.19	0.31	0.24
12	0.18	0.28	0.22
13	0.23	0.06	0.09
14	0.72	0.71	0.72
15	0.69	0.81	0.74
16	0.44	0.26	0.32
17	0.18	0.41	0.25
<b>Micro Average</b>	0.43	0.43	0.42
<b>Macro Average</b>	0.43	0.42	0.42

The confusion matrix in Figure 8.5 shows the correctly and incorrectly classified devices that that this model has predicted. It can be shown that the there are many devices have been yet falsely classified as the model still needs to be trained for a greater number of epochs. This concludes that using the binary input instead, the model is not reliable in performing the classification between the devices.

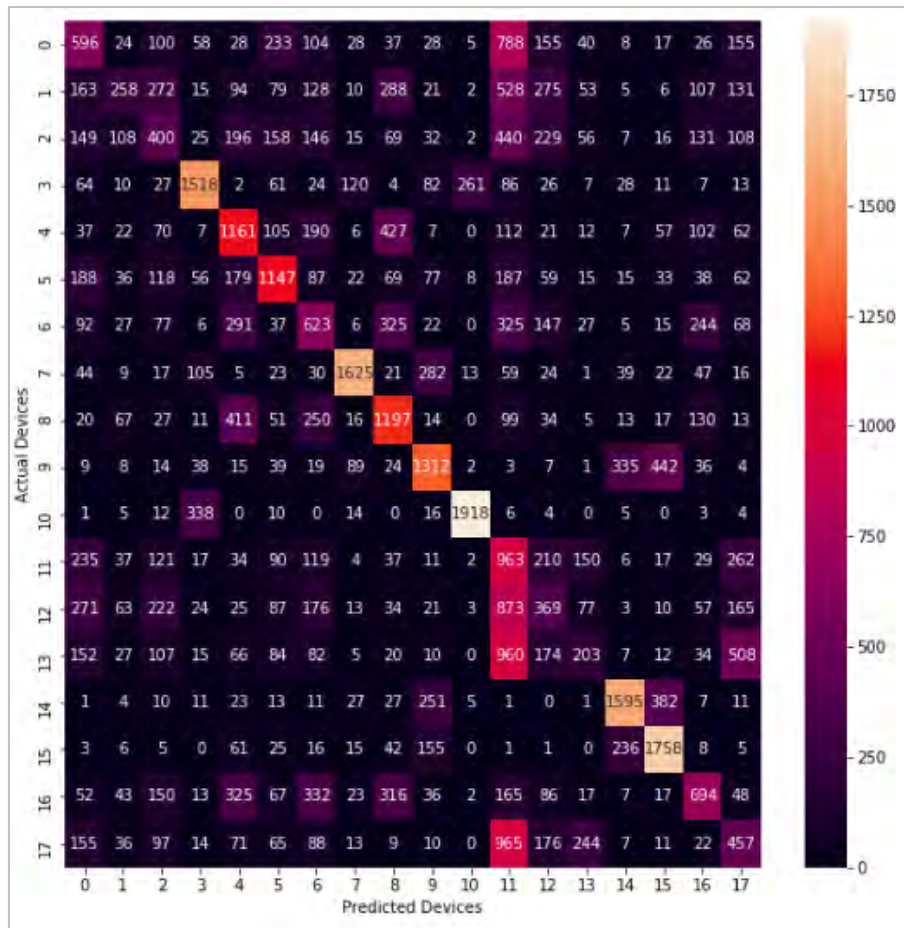


Figure 8.5: Confusion Matrix for CNN (Binary Input).

The main difference into why the work in [71] was able to achieve a higher accuracy with a smaller number of epochs is because each of the devices is different, and a model has been built for each of the device type.

## Chapter 9. Discussion of Model Results

This chapter summarizes the models built in this thesis and highlights the best results amongst them. In section 9.1, the results for each of the traditional models and the CNN models are presented. We also compare the best performing traditional model in comparison with our model, the CNN.

### 9.1. Summary of Model Results

This section presents the K-Fold results in terms of the mean and standard deviation. The standard deviation of the 10 folds run is shown in Table 9.1.

Table 9.1: Results for all Models on K-Fold – Mean (Standard Deviation)

Model Name	Accuracy	Precision	Recall	F1-score
<b>CNN Exp 2</b>	0.4430 (0.0014)	0.1547 (0.0042)	0.235 (0.0032)	0.1356 (0.0019)
<b>Bayesian</b>	0.2196 (0.0024)	0.1727 (0.0035)	0.2194 (0.0023)	0.1683 (0.0016)
<b>SVM Linear</b>	0.4835 (0.0052)	0.5035 (0.0075)	0.482 (0.0051)	0.4546 (0.0035)
<b>SVM Non-Linear</b>	0.8233 (0.0032)	0.8346 (0.0036)	0.8232 (0.0032)	0.8221 (0.0031)
<b>XG-Boost</b>	0.8652 (0.0050)	0.8713 (0.0047)	0.8651 (0.0050)	0.8650 (0.0051)
<b>Gradient Boost</b>	0.8819 (0.0026)	0.8898 (0.0036)	0.8818 (0.0027)	0.8819 (0.0027)
<b>LightGBM</b>	0.9241 (0.0025)	0.9248 (0.0026)	0.9241 (0.0025)	0.9238 (0.0026)
<b>Random Forest</b>	0.9501 (0.0036)	0.9505 (0.0036)	0.9501 (0.0036)	0.9501 (0.0036)
<b>CNN – Exp 1</b>	0.9993 (0.0001)	0.9900 (0.0063)	0.9971 (0.0020)	0.9959 (0.0012)

Statistical tests (Kruskal-Wallis followed by a Pairwise Wilcoxon tests) were also performed to check whether the models were statistically different. Kruskal-Wallis tests showed statistically significant difference ( $\chi^2=84.192$ ,  $p < 0.05$ ) exists between the models exists. Then, a Pairwise Wilcoxon test was performed to check for models that were statistically similar. The result of this test is shown in Table 9.2. It can be shown that CNN model before and after tuning are similar since the value was greater than 0.05, and the SVM linear model and the CNN (experiment 2) are similar too.

Table 9.2: Pairwise Wilcoxon Test Results (p-values)

	Bayesian	CNN (Exp 1)	CNN (Exp 2)	Gradient Boost	Light GBM	CNN (no tuning)	RF	SVM Linear	SVM RBF
CNN (Exp 1)	0.0081	-	-	-	-	-	-	-	-
CNN (Exp 2)	0.1253	0.0079	-	-	-	-	-	-	-
Gradient Boost	0.0004	0.0081	0.0080	-	-	-	-	-	-
Light GBM	0.0004	0.0081	0.0080	0.0004	-	-	-	-	-
CNN (With no tuning)	0.0080	1	0.0078	0.0080	0.0080	-	-	-	-
Random Forest	0.0081	0.0081	0.0079	0.0081	0.0081	0.0079	-	-	-
SVM Linear	0.0004	0.0081	0.0080	0.0004	0.0004	0.0080	0.008	-	-
SVM RBF	0.0004	0.0081	0.0080	0.0004	0.0004	0.0080	0.008	0.0004	-
XG-Boost	0.0004	0.0081	0.0080	0.0004	0.0004	0.0080	0.008	0.0004	0.0004

The work in this thesis has also shown that the method followed in this thesis in leveraging a single network feature to identify between identical devices could more cost effective and realistic than the commonly used PUF-based authentication methods on the resource constrained devices. PUF-based methods have their own weaknesses and limitations. For example, the SRAM-PUF method that is commonly used relies on the behaviour of the SRAM memory, assuming that the memory is always uninitialized [111]. Also, the memory-based PUF methods suffer from the size of the Challenge Response Pair increasing exponentially with their size [112] requiring additional memory. Nevertheless, some research has also shown that modelling attacks (that aim to mimic the way a PUF behaves) or some side channel attacks are possible on PUF [113]. Moreover, the operations involving PUF-based methods, such as extracting information from the hardware of the device, implementing cryptographic operations etc. makes the process slower, rather than only picking up the time between network packets in a traffic flow.

## Chapter 10. Conclusion and Future Work

The work in this research looks into addressing the problem of an IoT device being physically replaced by another illegitimate one. This is done by generating a unique device fingerprint from the data the devices generate. That is; given a smart IoT environment, features from the IoT devices are leveraged to create unique device fingerprints for each of them. In this thesis, this is achieved by collecting data from 18 edge devices and selecting a specific feature to study. The feature studied in this thesis is the Inter-Arrival time between network packets. Then, a CNN was built and trained to identify the device. The experiments are performed on widely used IoT edge device ESP32 running the MQTT protocol. The work is also compared against widely known machine learning algorithms, Random Forest, Bayesian, Gradient Boost, LightGBM, SVM, and XG-Boost. The experiment involved 18 ESP32 Devices running over a period of around 28 days. The results of the experiments showed that the CNN outperformed other models with an accuracy over 0.99, concluding that CNN can be leveraged to train the Inter-Arrival time feature from the devices to uniquely distinguish between them.

There are some limitations that we recognize in this thesis. In a typical IoT world, the number of connected devices is very huge. In this thesis, the experiment of device authentication was performed for 18 devices. However, it can suffice for a small smart environment such as a smart home that contain fewer number of IoT devices.

In the future, this research can be taken forward to experiment the problem using Recurrent Neural Networks (RNN) which is also a widely common class of neural networks. It can be compared to the results of the CNN. Also, other features can be leveraged other than the Inter-Arrival time, or a combination of features from the network, transmission, or application layer to train the model and study how they will perform in detecting between the devices. Finally, the model can be tested to see whether it will scale for huge number of devices.

## References

- [1] S. Gillis, "Internet of things (IoT)", [Online]. Available: <https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT>. [Accessed: Jun. 12, 2019].
- [2] T. Aziz and E. Haq, "Security Challenges Facing IoT Layers and its Protective Measures", *International Journal of Computer Applications*, vol. 179, no. 27, pp. 31-35, 2018. Available: 10.5120/ijca2018916607.
- [3] R. Irons, A. Sabella and M. Yannuzzi, "Communication Protocols for IoT > IoT and Security Standards and Best Practices | Cisco Press", *Ciscopress.com*, 2021. [Online]. Available: <https://www.ciscopress.com/articles/article.asp?p=2923211&seqNum=6>. [Accessed: Jun. 12, 2019].
- [4] G. Schatz, "The Complete List Of Wireless IoT Network Protocols", *Link-labs.com*, 2019. [Online]. Available: <https://www.link-labs.com/blog/complete-list-iot-network-protocols>. [Accessed: Sep. 19, 2020].
- [5] "Wi-Fi & Bluetooth MCUs and AIoT Solutions | Espressif Systems", *Espressif.com*, 2018. [Online]. Available: <https://www.espressif.com>. [Accessed: Jan. 13, 2019].
- [6] "ESP8266 Wi-Fi MCU | Espressif Systems", *Espressif.com*, 2018. [Online]. Available: <https://www.espressif.com/en/products/socs/esp8266>. [Accessed: Jan. 11, 2019].
- [7] "ESP8266 Thing Development Board Hookup Guide - learn.sparkfun.com.", [Online]. <https://learn.sparkfun.com/tutorials/esp8266-thing-development-board-hookup-guide/using-the-esp8266-in-arduino>. [Accessed Jan. 04, 2019].
- [8] "ESP32 Wi-Fi & Bluetooth MCU | Espressif Systems.", [Online]. Available: <https://www.espressif.com/en/products/socs/esp32>. [Accessed Jan. 04, 2019].
- [9] "ESP32 Thing Hookup Guide - learn.sparkfun.com.", [Online]. Available: <https://learn.sparkfun.com/tutorials/esp32-thing-hookup-guide>. [Accessed Jan. 04, 2019].
- [10] "RTOS | Espressif Systems.", [Online]. Available: <https://www.espressif.com/en/tags/rtos>. [Accessed Jan. 04, 2019].
- [11] "Understanding the MQTT Protocol Packet Structure.", [Online]. Available: <http://www.steves-internet-guide.com/mqtt-protocol-messages-overview/>. [Accessed Jan. 24, 2019].
- [12] "Get to Know MQTT: The Messaging Protocol for the Internet of Things.", [Online]. Available: <https://thenewstack.io/mqtt-protocol-iot/>.
- [13] J. Schmidhuber, "Deep Learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015, doi: 10.1016/j.neunet.2014.09.003.
- [14] "Basic CNN Architecture: Explaining 5 Layers of Convolutional Neural Network.", [Online]. Available: <https://www.upgrad.com/blog/basic-cnn-architecture/>. [Accessed Jan. 04, 2019].
- [15] "The Blurry Lines of Supervised and Unsupervised Learning | by Paul-Louis Pröve | Towards Data Science.", [Online]. Available: <https://towardsdatascience.com/the-blurry-lines-of-supervised-and-unsupervised-learning-b8a2aa04c8b0>. [Accessed Feb. 14, 2019].

- [16] M. El-Hajj, A. Fadlallah, M. Chamoun and A. Serhrouchni, "A survey of internet of things (IoT) authentication schemes," *Sensors (Switzerland)*, vol. 19, no. 5, pp. 1–43, 2019, doi: 10.3390/s19051141.
- [17] "Symmetric Key | HowStuffWorks.", [Online]. Available: <https://computer.howstuffworks.com/encryption2.htm> [Accessed Feb. 18, 2019].
- [18] "What is Asymmetric Cryptography and How Does it Work?", [Online]. Available: <https://searchsecurity.techtarget.com/definition/asymmetric-cryptography> [Accessed Feb. 08, 2019].
- [19] L. Zhou, X. Lu, K. Yeh, C. Su, and W. Chiu, "Lightweight IoT-based authentication scheme in cloud computing circumstance," *Futur. Gener. Comput. Syst.*, vol. 91, pp. 244–251, 2019, doi: 10.1016/j.future.2018.08.038.
- [20] F. Moghaddam, S. G. Moghaddam, S. Rouzbeh, S. K. Araghi, N. M. Alibeigi and S. D. Varnosfaderani, "A scalable and efficient user authentication scheme for cloud computing environments," *2014 IEEE REGION 10 SYMPOSIUM*, 2014, pp. 508-513, doi: 10.1109/TENCONSpring.2014.6863086.
- [21] M. Hammi, E. Livolant, P. Bellot, A. Serhrouchni and P. Minet, "A lightweight mutual authentication protocol for the IoT," *2017 iCatse International Conference on Mobile and Wireless Technology*, 2017, doi: ff10.1007/978-981-10-5281-1.
- [22] H. Hamidi, "An approach to develop the smart health using Internet of Things and authentication based on biometric technology," *Futur. Gener. Comput. Syst.*, vol. 91, pp. 434–449, 2019, doi: 10.1016/j.future.2018.09.024.
- [23] M. Alotaibi, "An enhanced symmetric cryptosystem and biometric-based anonymous user authentication and session key establishment scheme for WSN," *IEEE Access*, vol. 6, pp. 70072–70087, 2018, doi: 10.1109/ACCESS.2018.2880225.
- [24] A. Mukherjee, "Physical-Layer Security in the Internet of Things: Sensing and Communication Confidentiality Under Resource Constraints," in *Proceedings of the IEEE*, vol. 103, no. 10, pp. 1747-1761, Oct. 2015, doi: 10.1109/JPROC.2015.2466548.
- [25] "Challenge-Response Authentication - Zenfolio.", [Online]. Available: <https://www.zenfolio.com/zf/help/api/guide/auth/auth-challenge>. [Accessed Feb. 08, 2019].
- [26] T. Mick, R. Tourani and S. Misra, "LASER: Lightweight Authentication and Secured Routing for NDN IoT in Smart Cities," in *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 755-764, April 2018, doi: 10.1109/JIOT.2017.2725238.
- [27] P. Gope, J. Lee and T. Q. S. Quek, "Lightweight and Practical Anonymous Authentication Protocol for RFID Systems Using Physically Unclonable Functions," in *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 11, pp. 2831-2843, Nov. 2018, doi: 10.1109/TIFS.2018.2832849.
- [28] S. Park, S. Lim, D. Jeong, J. Lee, J. Yang and H. Lee, "PUFSec: Device fingerprint-based security architecture for Internet of Things," *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, 2017, pp. 1-9, doi: 10.1109/INFOCOM.2017.8057146.
- [29] J. Wallrabenstein, "Practical and Secure IoT Device Authentication Using Physical Unclonable Functions," *2016 IEEE 4th International Conference on*

- Future Internet of Things and Cloud (FiCloud)*, Vienna, Austria, 2016, pp. 99-106, doi: 10.1109/FiCloud.2016.22.
- [30] M. Barbareschi, P. Bagnasco and A. Mazzeo, "Authenticating IoT Devices with Physically Unclonable Functions Models," *2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, 2015, pp. 563-567, doi: 10.1109/3PGCIC.2015.117.
- [31] W. Liang, S. Xie, J. Long, K. Li, D. Zhang and K. Li, "A double PUF-based RFID identity authentication protocol in service-centric internet of things environments", *Information Sciences*, vol. 503, pp. 129-147, 2019. Available: 10.1016/j.ins.2019.06.047.
- [32] B. Srinivasu, P. Vikramkumar, A. Chattopadhyay and K. Lam, "CoLPUF : A Novel Configurable LFSR-based PUF," *2018 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, 2018, pp. 358-361, doi: 10.1109/APCCAS.2018.8605643.
- [33] H. Xu, J. Ding, P. Li, F. Zhu and R. Wang, "A lightweight rfid mutual authentication protocol based on physical unclonable function," *Sensors (Switzerland)*, vol. 18, no. 3, pp. 1–20, 2018, doi: 10.3390/s18030760.
- [34] "What is RFID and How Does it Work?," [Online]. Available: <https://www.abr.com/what-is-rfid-how-does-rfid-work>. [Accessed Apr. 18, 2019].
- [35] F. Jaafar, "An Integrated Architecture for IoT Fingerprinting," *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, Prague, Czech Republic, 2017, pp. 601-602, doi: 10.1109/QRS-C.2017.112.
- [36] A. Brian and L. Arockiam, "An IOT based secured smart library system with NFC based book tracking," *International Journal of Emerging Technology and Computer Science & Electronics*, 2014, vol. 11, pp. 18–21, 2014.
- [37] T. Goethem, W. Scheepers, D. Preuveneers and W. Joosen, "Accelerometer-based device fingerprinting for multi-factor mobile authentication," *Lect. Notes Comput. Sci.*, vol. 9639, pp. 106–121, 2016, doi: 10.1007/978-3-319-30806-7\_7.
- [38] B. Charyyev and M. H. Gunes, "Locality-Sensitive IoT Network Traffic Fingerprinting for Device Identification," in *IEEE Internet of Things Journal*, vol. 8, no. 3, pp. 1272-1281, 1 Feb.1, 2021, doi: 10.1109/JIOT.2020.3035087.
- [39] J. JORDAN, "Introduction to autoencoders," 2019. [Online]. Available: <https://www.jeremyjordan.me/autoencoders>. [Accessed Feb. 08, 2020].
- [40] M. Stewart, "Comprehensive Introduction to Autoencoders | by Matthew Stewart, PhD Researcher | Towards Data Science," 2019. [Online]. Available: <https://towardsdatascience.com/generating-images-with-autoencoders-77fd3a8dd368> [Accessed Apr. 08, 2020].
- [41] R. Hata, H. Akhand and K. Murase, "Multi-Valued Autoencoders and Classification of Large-Scale Multi-Class Problem," *Int. J. Adv. Comput. Sci. Appl.*, vol. 8, no. 11, pp. 19–26, 2017, doi: 10.14569/ijacsa.2017.081103.
- [42] G. Cavallar, L. Sampaio and M. Antonelli, "Unsupervised Representation Learning Using Convolutional and Stacked Auto-Encoders: A Domain and Cross-Domain Feature Space Analysis," *2018 31st SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, 2018, pp. 440-446, doi: 10.1109/SIBGRAPI.2018.00063.

- [43] S. Bhattamishra, "Deep probabilistic NMF using denoising autoencoders," *Int. J. Mach. Learn. Comput.*, vol. 8, no. 1, pp. 49–53, 2018, doi: 10.18178/ijmlc.2018.8.1.662.
- [44] A. Krizhevsky and E. Hinton., "Very Deep Autoencoders for Content-Based Image Retrieval," *ESANN 2011, 19th European Symposium on Artificial Neural Networks, Bruges, Belgium*, April 27-29, 2011, pp. 489–494.
- [45] "Deep-convolutional and Autoencoders.", [Online]. Available: <https://hackernoon.com/a-deep-convolutional-denoising-autoencoder-for-image-classification> [Accessed Feb. 08, 2019].
- [46] S. Leroux, S. Bohez, P. Maenhaut, N. Meheus, P. Simoens and B. Dhoedt, "Fingerprinting encrypted network traffic types using machine learning," *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, 2018, pp. 1-5, doi: 10.1109/NOMS.2018.8406218.
- [47] J. M. Espín, R. Font, J. G. Marín-Blazquez and F. Esquembre, "LOGICAL ACCESS ATTACKS DETECTION THROUGH AUDIO FINGERPRINTING IN AUTOMATIC SPEAKER VERIFICATION," *2018 IEEE 28th International Workshop on Machine Learning for Signal Processing (MLSP)*, 2018, pp. 1-6, doi: 10.1109/MLSP.2018.8517013.
- [48] G. Aceto, D. Ciuonzo, A. Montieri and A. Pescapé, "Traffic Classification of Mobile Apps through Multi-Classification," *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, 2017, pp. 1-6, doi: 10.1109/GLOCOM.2017.8254059.
- [49] F. Shaikh, E. Bou-Harb, J. Crichigno and N. Ghani, "A Machine Learning Model for Classifying Unsolicited IoT Devices by Observing Network Telescopes," *2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)*, 2018, pp. 938-943, doi: 10.1109/IWCMC.2018.8450404.
- [50] J. Park, H. Mun and Y. Lee, "Improving Tor Hidden Service Crawler Performance," *2018 IEEE Conference on Dependable and Secure Computing (DSC)*, 2018, pp. 1-8, doi: 10.1109/DESEC.2018.8625103.
- [51] Y. Meida, M. Bohadana, A. Shabtai, M. Ochoa, N. Tippenhauer, J. Guarnizo and Y. Elovici., "Detection of Unauthorized IoT Devices Using Machine Learning Techniques," arXiv, 2017.
- [52] V. Brik, S. Banerjee, M. Gruteser and S. Oh, "Wireless device identification with radiometric signatures," *Proc. Annu. Int. Conf. Mob. Comput. Networking, MOBICOM, 2008*, pp. 116–127, doi: 10.1145/1409944.1409959.
- [53] A. Hameed and A. Leivadreas, "IoT Traffic Multi-Classification Using Network and Statistical Features in a Smart Environment," *2020 IEEE 25th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, 2020, pp. 1-7, doi: 10.1109/CAMAD50429.2020.9209311.
- [54] A. Hussain and G. Oligeri, "The dark (and bright) side of IoT: Attacks and countermeasures to identification of smart home devices and services," arXiv, vol. 1, pp. 1–15, 2020, doi: 10.1007/978-3-030-68884-4\_10.
- [55] M. R. Shahid, G. Blanc, Z. Zhang and H. Debar, "IoT Devices Recognition Through Network Traffic Analysis," *2018 IEEE International Conference on Big Data (Big Data)*, 2018, pp. 5187-5192, doi: 10.1109/BigData.2018.8622243.

- [56] S. Sciancalepore, O. Ibrahim, G. Oligeri and R. Pietro, "PiNcH: An effective, efficient, and robust solution to drone detection via network traffic analysis," *Comput. Networks*, vol. 168, p. 107044, 2020, doi: 10.1016/j.comnet.2019.107044.
- [57] L. Chaddad, A. Chehab, I. Elhajj and A. Kayssi "AdaptiveMutate: a technique for privacy preservation", *Digital Communications and Networks*, vol. 5, no. 4, pp. 245-255, 2019. Available: 10.1016/j.dcan.2019.09.002.
- [58] L. Babun, H. Aksu, L. Ryan, K. Akkaya, E. S. Bentley and A. S. Uluagac, "Z-IoT: Passive Device-class Fingerprinting of ZigBee and Z-Wave IoT Devices," *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1-7, doi: 10.1109/ICC40277.2020.9149285.
- [59] G. Vaidya, A. Nambi, T. V. Prabhakar, V. Kumar T and S. Sudhakara, "IoT-ID: A Novel Device-Specific Identifier Based on Unique Hardware Fingerprints," *2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI)*, 2020, pp. 189-202, doi: 10.1109/IoTDI49375.2020.00026.
- [60] V. Rimmer, D. Preuveneers, M. Juarez, T. Goethem and W. Joosen, "Automated website fingerprinting through deep learning," *Network and Distributed System Security Symposium*, vol. 4, pp. 30-42, 2017, doi: 10.14722/ndss.2018.23105.
- [61] Y. Sun, X. Wang and X. Tang, "Deep Learning Face Representation from Predicting 10,000 Classes," *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1891-1898, doi: 10.1109/CVPR.2014.244.
- [62] X. Qiu, Z. Lit, X. Sun and T. Xu, "A Lightweight Intelligent Authentication Approach for Intrusion Detection," *2020 IEEE 31st Annual International Symposium on Personal, Indoor and Mobile Radio Communications*, 2020, pp. 1-6, doi: 10.1109/PIMRC48278.2020.9217112.
- [63] K. Yang, Q. Li and L. Sun, "Towards automatic fingerprinting of IoT devices in the cyberspace," *Comput. Networks*, vol. 148, pp. 318–327, 2019, doi: 10.1016/j.comnet.2018.11.013.
- [64] M. Skowron, A. Janicki and W. Mazurczyk, "Traffic Fingerprinting Attacks on Internet of Things Using Machine Learning," in *IEEE Access*, vol. 8, pp. 20386-20400, 2020, doi: 10.1109/ACCESS.2020.2969015.
- [65] J. Kotak and Y. Elovici., "IoT Device Identification Using Deep Learning", *13th International Conference on Computational Intelligence in Security for Information Systems (CISIS 2020)*, pp. 76-86, 2020. Available: 10.1007/978-3-030-57805-3\_8.
- [66] S. Aneja, N. Aneja and M. S. Islam, "IoT Device Fingerprint using Deep Learning," *2018 IEEE International Conference on Internet of Things and Intelligence System (IOTAIS)*, 2018, pp. 174-179, doi: 10.1109/IOTAIS.2018.8600824.
- [67] K. A. Simpson, R. Cziva and D. P. Pezaros, "Seiðr: Dataplane Assisted Flow Classification Using ML," *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, 2020, pp. 1-6, doi: 10.1109/GLOBECOM42002.2020.9348063.
- [68] O. Salman, I. Elhajj, A. Kayssi and A. Chehab, "Data representation for CNN based internet traffic classification: a comparative study", *Multimedia Tools and Applications*, vol. 12, pp. 7-19, 2020. Available: 10.1007/s11042-020-09459-4.
- [69] V. Thangavelu, D. M. Divakaran, R. Sairam, S. S. Bhunia and M. Gurusamy, "DEFT: A Distributed IoT Fingerprinting Technique," in *IEEE Internet of*

- Things Journal*, vol. 6, no. 1, pp. 940-952, Feb. 2019, doi: 10.1109/JIOT.2018.2865604.
- [70] K. Merchant, S. Revay, G. Stantchev and B. Nousain, "Deep Learning for RF Device Fingerprinting in Cognitive Communication Networks," in *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 160-167, Feb. 2018, doi: 10.1109/JSTSP.2018.2796446.
- [71] G. Spanos, K. Giannoutakis, K. Votis, B. Viano, and J. Gonzalez, "A Lightweight Cyber-Security Defense Framework for Smart Homes," *2020 International Conference on Innovations in Intelligent SysTems and Applications (INISTA)*, 2020, pp. 1-7, doi: 10.1109/INISTA49547.2020.9194689.
- [72] "DBSCAN Clustering Algorithm in Machine Learning." [Online]. Available: <https://www.kdnuggets.com/2020/04/dbscan-clustering-algorithm-machine-learning.html> [Accessed Apr. 18, 2021].
- [73] G. Qing, H. Wang, L. Guo and J. Yang, "Device Type Identification via Network Traffic and Lightweight Convolutional Neural Network for Internet of Things," in *IEEE Access*, vol. 8, pp. 200219-200228, 2020, doi: 10.1109/ACCESS.2020.3032469.
- [74] "MQTT: The Standard for IoT Messaging." [Online]. Available: <https://www.mqtt.org/> [Accessed Apr. 18, 2021].
- [75] "PCAP: Packet Capture, what it is & what you need to know." [Online]. Available: <https://www.comparitech.com/net-admin/pcap-guide/> [Accessed Apr. 18, 2021].
- [76] "Wireshark · Go Deep." [Online]. Available: <https://www.wireshark.org/> [Accessed Jan. 23, 2021].
- [77] B. Suhas et al., "Speech task based automatic classification of ALS and Parkinson's Disease and their severity using log Mel spectrograms," *2020 International Conference on Signal Processing and Communications (SPCOM)*, 2020, pp. 1-5, doi: 10.1109/SPCOM50965.2020.9179503.
- [78] J. Lyons, "Practical Cryptography," 2013. [Online]. Available: <http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/> [Accessed Nov. 30, 2020].
- [79] "Classification: True vs. False and Positive vs. Negative," 2020. [Online]. Available: <https://developers.google.com/machine-learning/crash-course/classification/true-false-positive-negative> [Accessed Feb. 23, 2021].
- [80] S. Narkhede, "Understanding Confusion Matrix," 2018. [Online]. Available: <https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62> [Accessed Feb. 23, 2021].
- [81] S. Minaee, "20 Popular Machine Learning Metrics. Part 1: Classification & Regression Evaluation Metrics," 2019. [Online]. Available: <https://towardsdatascience.com/20-popular-machine-learning-metrics-part-1-classification-regression-evaluation-metrics-1ca3e282a2ce> [Accessed Mar. 23, 2021].
- [82] "Classification: Accuracy," 2020. [Online]. Available: <https://developers.google.com/machine-learning/crash-course/classification/accuracy> [Accessed Feb. 23, 2021].
- [83] "Classification: Precision and Recall," 2020. [Online]. Available: <https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall> [Accessed Feb. 23, 2021].

- [84] K. Shung, "Accuracy, Precision, Recall or F1?," 2018. [Online]. Available: <https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9> [Accessed Mar. 23, 2021].
- [85] "Classification: ROC Curve and AUC ," 2020. [Online]. Available: <https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc> [Accessed Feb. 27, 2021].
- [86] N. Sarang, "Understanding AUC - ROC Curve," 2018. [Online]. Available: <https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5> [Accessed Mar. 23, 2021].
- [87] A. AL-Masri, "What Are Overfitting and Underfitting in Machine Learning?," 2019. [Online]. Available: <https://towardsdatascience.com/what-are-overfitting-and-underfitting-in-machine-learning-a96b30864690> [Accessed Mar. 23, 2021].
- [88] T. Wong and P. Yeh, "Reliable Accuracy Estimates from k-Fold Cross Validation," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 8, pp. 1586-1594, 1 Aug. 2020, doi: 10.1109/TKDE.2019.2912815.
- [89] M. Peng, Z. Wu, Z. Zhang and T. Chen, "From Macro to Micro Expression Recognition: Deep Learning on Small Datasets Using Transfer Learning," *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)*, 2018, pp. 657-661, doi: 10.1109/FG.2018.00103.
- [90] "Micro- and Macro-average of Precision, Recall and F-Score," 2018. [Online]. Available: <https://tomaxent.com/2018/04/27/Micro-and-Macro-average-of-Precision-Recall-and-F-Score/> [Accessed Mar. 23, 2021].
- [91] "Pcap File - an overview." 2008. [Online]. Available: <https://www.sciencedirect.com/topics/computer-science/pcap-file> [Accessed Feb. 04, 2021].
- [92] W. Koehrsen, "An Implementation and Explanation of the Random Forest in Python," 2018. [Online]. Available: <https://towardsdatascience.com/an-implementation-and-explanation-of-the-random-forest-in-python> [Accessed Feb. 03, 2021].
- [93] "Machine Learning Random Forest Algorithm - Javatpoint." [Online]. Available: <https://www.javatpoint.com/machine-learning-random-forest-algorithm> [Accessed Jan. 23, 2021].
- [94] "Bayesian Model - an overview | ScienceDirect Topics." [Online]. Available: <https://www.sciencedirect.com/topics/computer-science/bayesian-model> [Accessed Feb. 03, 2021].
- [95] M. Rashed-Al-Mahfuz, M. R. Hoque, B. K. Pramanik, M. E. Hamid and M. A. Moni, "SVM Model for Feature Selection to Increase Accuracy and Reduce False Positive Rate in Falls Detection," *2019 International Conference on Computer, Communication, Chemical, Materials and Electronic Engineering (IC4ME2)*, 2019, pp. 1-5, doi: 10.1109/IC4ME247184.2019.9036529.
- [96] J. Brownlee, "How to Develop a Light Gradient Boosted Machine (LightGBM) Ensemble," 2020. [Online]. Available: <https://machinelearningmastery.com/light-gradient-boosted-machine-lightgbm-ensemble/> [Accessed Feb. 23, 2021].
- [97] H. Liu et al., "LightGBM-Based Prediction of Remaining Useful Life for Electric Vehicle Battery under Driving Conditions," *2020 IEEE Sustainable Power and Energy Conference (iSPEC)*, 2020, pp. 2577-2582, doi: 10.1109/iSPEC50848.2020.9351029.

- [98] S. Luo and T. Chen, "Two Derivative Algorithms of Gradient Boosting Decision Tree for Silicon Content in Blast Furnace System Prediction," in *IEEE Access*, vol. 8, pp. 196112-196122, 2020, doi: 10.1109/ACCESS.2020.3034566.
- [99] J. Brownlee, "A Gentle Introduction to the Gradient Boosting Algorithm for Machine Learning," 2016. [Online]. Available: <https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/> [Accessed Jan. 18, 2021].
- [100] "A simple example of visualizing gradient boosting." [Online]. Available: [https://www.researchgate.net/figure/A-simple-example-of-visualizing-gradient-boosting\\_fig5](https://www.researchgate.net/figure/A-simple-example-of-visualizing-gradient-boosting_fig5) [Accessed Jan. 18, 2021].
- [101] H. Li, Y. Cao, S. Li, J. Zhao and Y. Sun, "XGBoost Model and Its Application to Personal Credit Evaluation," in *IEEE Intelligent Systems*, vol. 35, no. 3, pp. 52-61, 1 May-June 2020, doi: 10.1109/MIS.2020.2972533.
- [102] O. Köpüklü, M. Babae, S. Hörmann and G. Rigoll, "Convolutional Neural Networks with Layer Reuse," *2019 IEEE International Conference on Image Processing (ICIP)*, 2019, pp. 345-349, doi: 10.1109/ICIP.2019.8802998.
- [103] T. Pala, U. Güvenç, H. T. Kahraman, İ. Yücedağ and Y. Sönmez, "Comparison of Pooling Methods for Handwritten Digit Recognition Problem," *2018 International Conference on Artificial Intelligence and Data Processing (IDAP)*, 2018, pp. 1-5, doi: 10.1109/IDAP.2018.8620848.
- [104] Arc, "Convolutional Neural Networks," 2018. [Online]. Available: <https://towardsdatascience.com/convolutional-neural-network-17fb77e76c05> [Accessed Jan. 03, 2021].
- [105] D. Muttaqin and S. Suyanto, "Speech Emotion Detection Using Mel-Frequency Cepstral Coefficient and Hidden Markov Model," *2020 3rd International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, 2020, pp. 463-466, doi: 10.1109/ISRITI51436.2020.9315433.
- [106] H. Fayek, "Speech Processing for Machine Learning: Filter banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between," 2016. [Online]. Available: <https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html> [Accessed Mar. 23, 2021].
- [107] R. R. Subramanian, N. Akshith, G. N. Murthy, M. Vikas, S. Amara and K. Balaji, "A Survey on Sentiment Analysis," *2021 11th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, 2021, pp. 70-75, doi: 10.1109/Confluence51648.2021.9377136.
- [108] S. Wang, M. Li, E. Zhu, J. Hu and X. Liu, "K-Means Clustering With Incomplete Data," in *IEEE Access*, vol. 7, pp. 69162-69171, 2019, doi: 10.1109/ACCESS.2019.2910287.
- [109] W. Wan et al., "Information Entropy Based Feature Pooling for Convolutional Neural Networks," *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 3404-3413, doi: 10.1109/ICCV.2019.00350.
- [110] J. Brownlee, "How to use Learning Curves to Diagnose Machine Learning Model Performance," 2019. [Online]. Available: <https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/> [Accessed Jan. 23, 2021].
- [111] S. Eiroa, I. Baturone and A. Acosta, "Using Physical Unclonable Functions for Hardware Authentication: A Survey", *Computer Networks*, vol. 183, p. 232, 2020. Available: <https://doi.org/10.1016/j.comnet.2020.107593>

- [112] H. Ning, F. Farha A. Ullah and L. Mao, "Physical unclonable function: Architectures, applications and challenges for dependable security," *IET Circuits, Devices and Systems*, vol. 14, no. 4, pp. 407–424, 2020, doi: 10.1049/iet-cds.2019.0175.
- [113] A. Shamsoshoara, A. Korenda, F. Afghah and S. Zeadally, "A survey on physical unclonable function (PUF)-based security solutions for Internet of Things", *Computer Networks*, vol. 183, p. 107593, 2020. Available: 10.1016/j.comnet.2020.107593.

## Appendix A

This appendix contains the code of the data preprocessing in Chapter 4.

### Sampling Parameters:

```
Fs = 10000; % Sampling Frequency
nfft = 512; % NFFT
N = 512; % number of samples per frame
total = 0;
x = 1;
windowSize = 512; % number of samples per window

% Read the values
Xaxis = S(1:10240,[1]); % counter of 0.1 ms
Yaxis = S(1:10240,[2]); % IAT
```

Figure A.1: Sampling Parameters.

### Mel Parameters:

- **Lowmelfreq:**  $1125 \cdot \log(1 + (50/700)) - 50$  is the low frequency in Hz
- **Highmelfreq:**  $1125 \cdot \log(1 + (5000/700)) - 5000$  is the high frequency in Hz
- **noOfBanks:** 23
- **minverse:**  $700 \cdot (\exp(\text{mspace}/1125) - 1)$

```
lowmelfreq = 1125*log(1+(50/700)); % Low Freq as 100 Hz
highmelfreq = 1125*log(1+(5000/700)); % High Freq as 10000 Hz (less than half the sample rate)
noOfBanks = 23;
mspace = linspace(lowmelfreq, highmelfreq, 25); % Create even spaced values

minverse = 700*(exp(mspace/1125)-1); % Convert Mel back to Hz
bin = floor((nfft+1)*minverse/Fs); % Fix the resolution
```

Figure A.2: MelSpec Parameters.

### Framing:

```
% ----- STEP 1: Frame signals (windowing)
|
|
| for col = 1:noOfFrames
|   for row = 1>windowSize
|     framedsignal(:,col) = Yaxis(x:x>windowSize-1);
|     end
|     x = x>windowSize-1;
|   end
| % ----- End of framing (windowing) -----
```

Figure A.3: Framing Signal.

### FFT of each frame:

```
% STEP 2A: Take FFT of each frame|  
  
% FFT Signal  
fftsignal = fft(framedsignal);  
fftsignal = abs(fftsignal);
```

Figure A.4: Frame FFT.

### Periodogram for each frame, and keep only the first half of the coefficients:

```
% % STEP 2B: Take the Periodogram of each of the above fft-ed frames  
  
Periodogram = abs(fftsignal);  
Periodogram = Periodogram.^2;  
Periodogram = Periodogram/nfft;
```

Figure A.5: Periodogram of Frames.

### Mel Filter Banks, initially with 23 banks, and only 13 will be kept later:

```
for m = 2:noOfBanks+1  
    f_m_minus = (bin(m-1));  
    f_m = (bin(m));  
    f_m_plus = (bin(m+1));  
  
    for k = f_m_minus:f_m  
        fbank(m-1,k) = (k - bin(m-1))/(bin(m) - bin(m-1));  
    end  
    for k = f_m:f_m_plus  
        fbank(m-1,k) = (bin(m+1) - k)/(bin(m+1) - bin(m));  
    end  
end  
fbank = fbank'
```

Figure A.6: MelFilter Banks.

### Apply the filter banks to each of the frames, and get sum of coefficients:

```
% STEP 3B: Apply Mel Banks to periodogram and get sum of coefficients
%result = zeros(nfft/2,noOfBanks);
x = 1
for frame = 1:noOfFrames

    temp = halfedperiodogram(:,frame);

    result = zeros(nfft/2,noOfBanks);

    for i=1:noOfBanks
        result(:,i) = temp.*fbank(:,i);
    end

    % bankpower(:,x:x+9) = result(:,1:noOfBanks);
    result2 = sum(result);

    % STEP 3c: add coeff:
    sumofcoeff(:,frame) = result2';
    x = x+10

end

x = 1;
```

Figure A.7: Applying Filterbanks to Frames.

### Log of the results:

```
% % STEP 4: Take the log:
logofcoeff =log(sumofcoeff);
```

Figure A.8: Log of Mel Coefficient.

### DCT of the above results:

```
% STEP 5: Get the DCT:
dctofcoeff = dct(logofcoeff);
```

Figure A.9: DCT of Log Coefficient.

## Appendix B

This Appendix contains the Mel plots for all the devices explained in Chapter 7.

### Device 2: 192.168.0.111

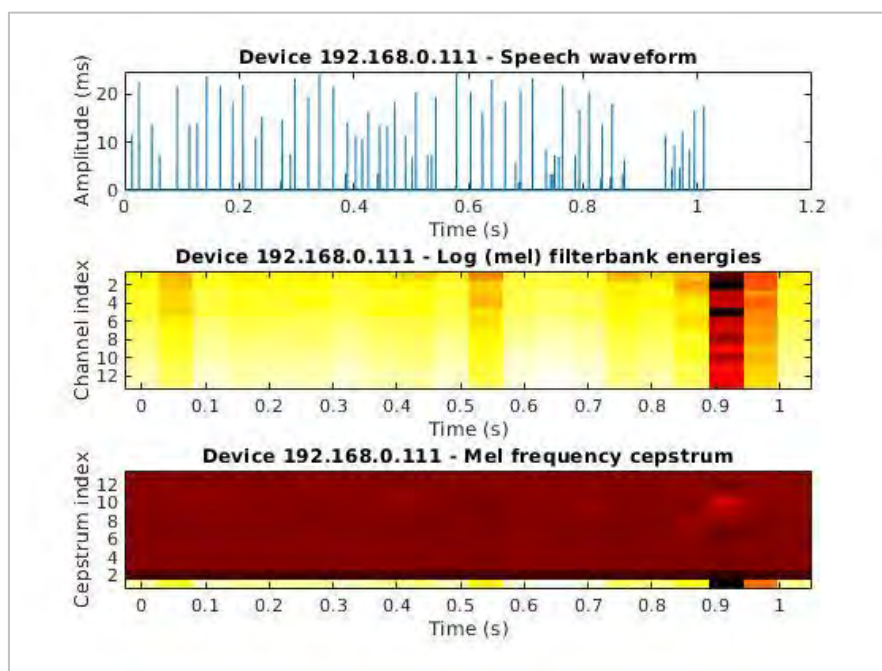


Figure B.1: Device 2 Time and MelPlots.

### Device 3: 192.168.0.122

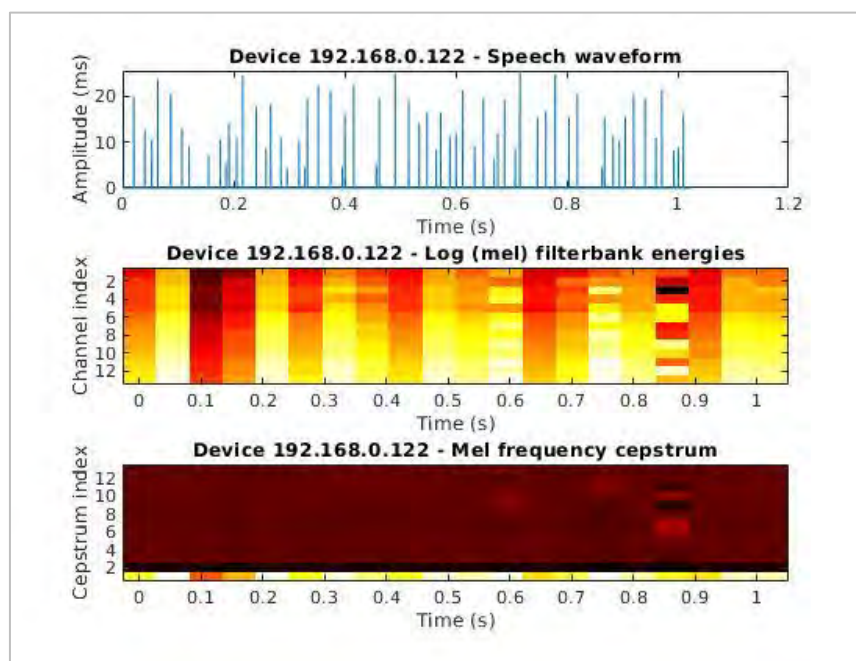


Figure B.2: Device 3 Time and MelPlots.

### Device 4: 192.168.0.124

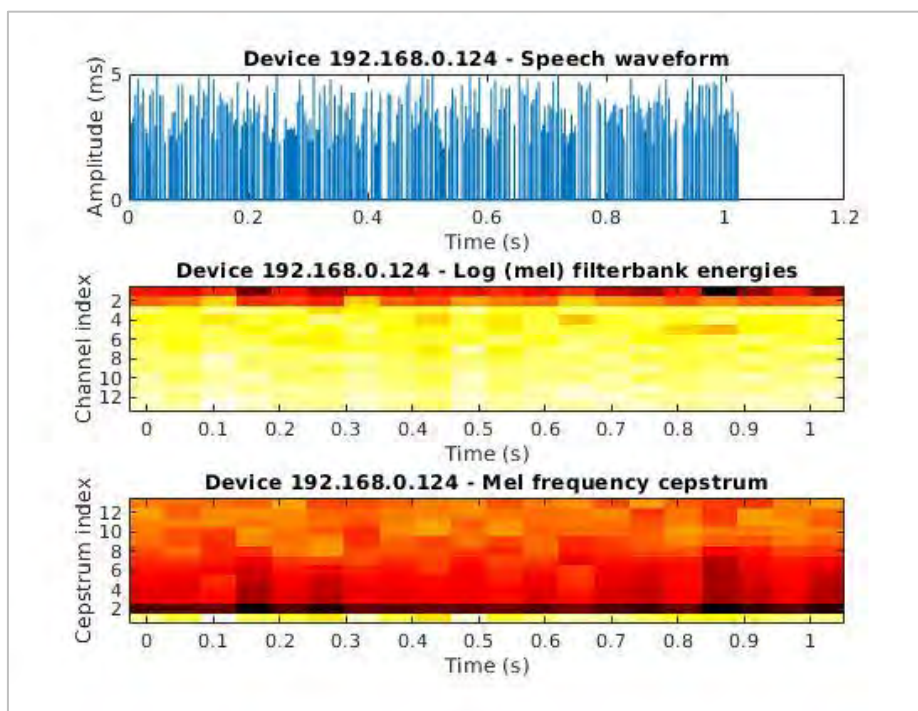


Figure B.3: Device 4 Time and MelPlots.

### Device 5: 192.168.0.125

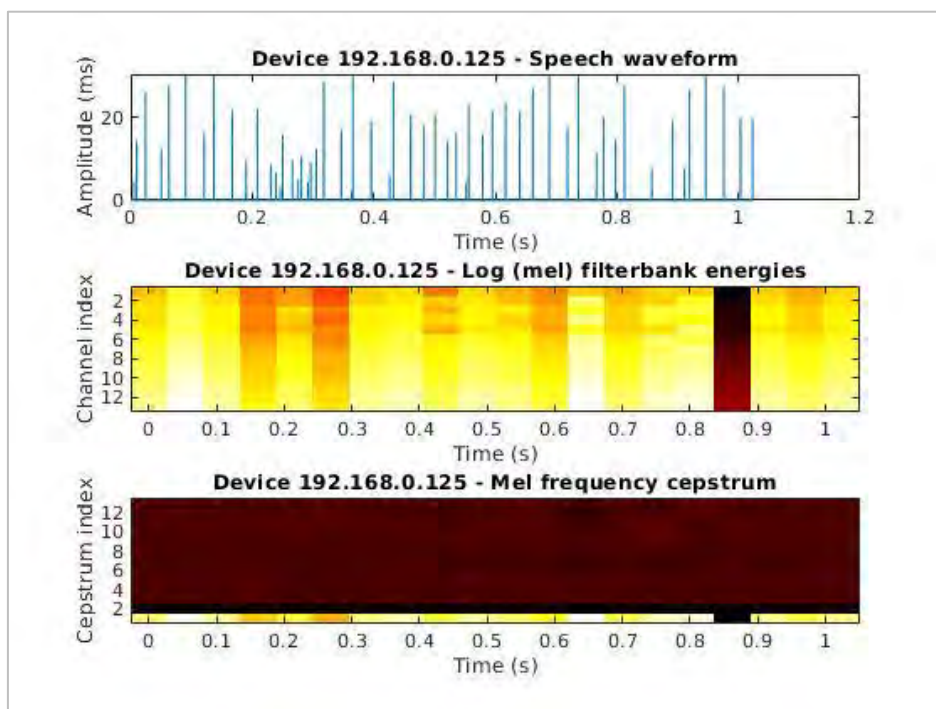


Figure B.4: Device 5 Time and MelPlots.

**Device 6: 192.168.0.126**

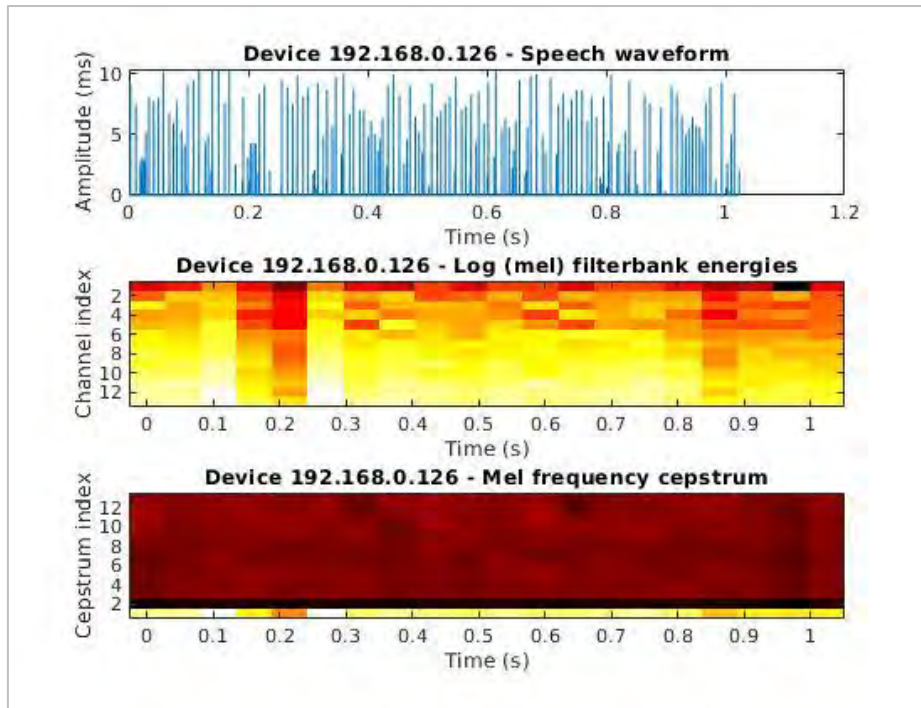


Figure B.5: Device 6 Time and MelPlots.

**Device 7: 192.168.0.128**

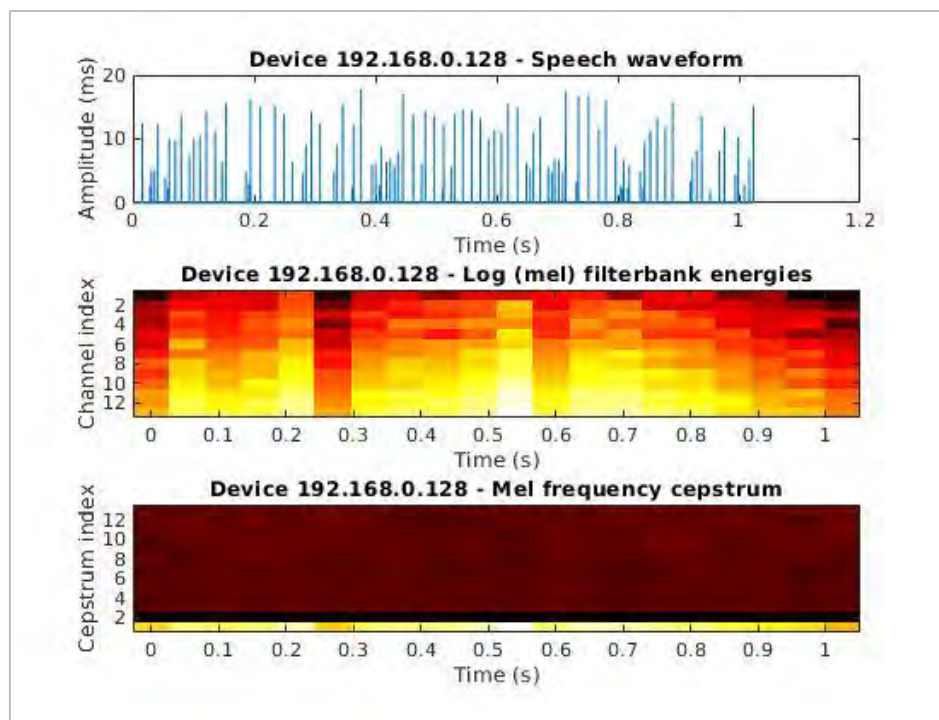


Figure B.6: Device 7 Time and MelPlots.

### Device 8: 192.168.0.129

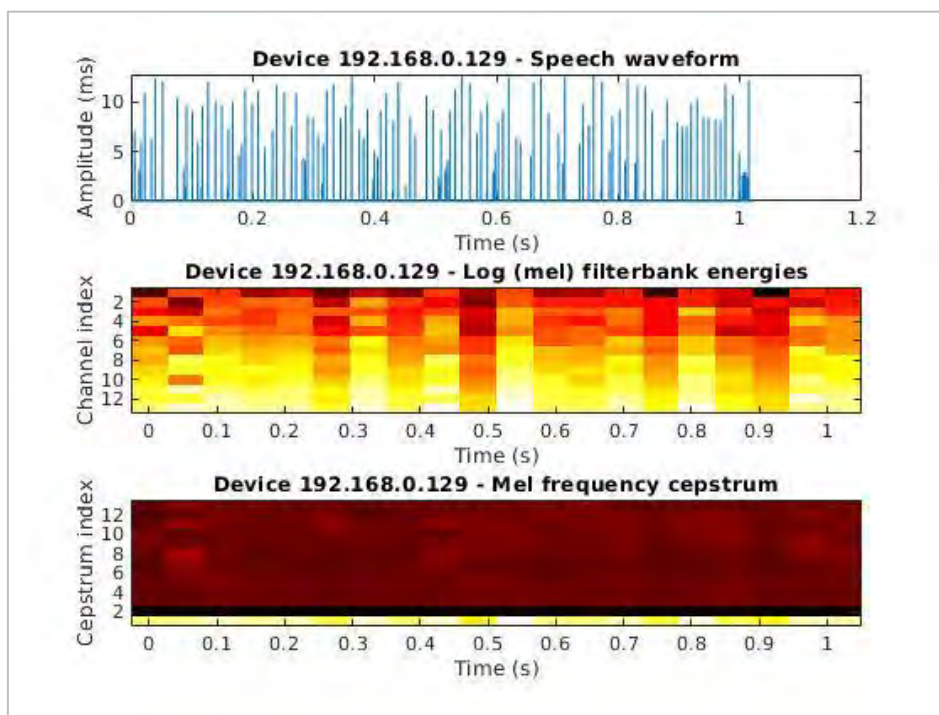


Figure B.7: Device 8 Time and MelPlots.

### Device 9: 192.168.0.130

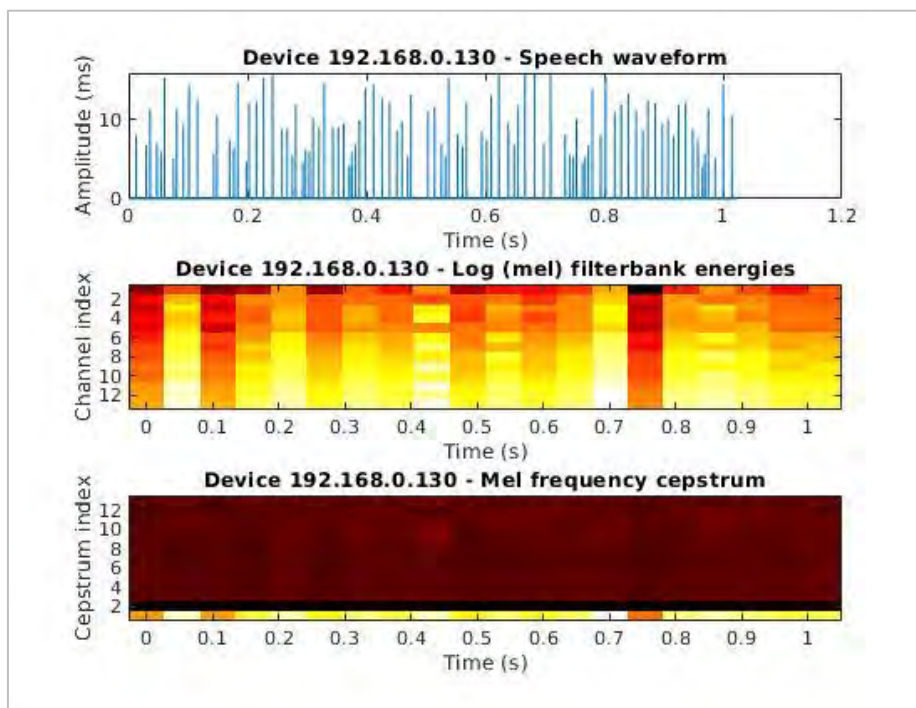


Figure B.8: Device 9 Time and MelPlots.

### Device 10: 192.168.0.132

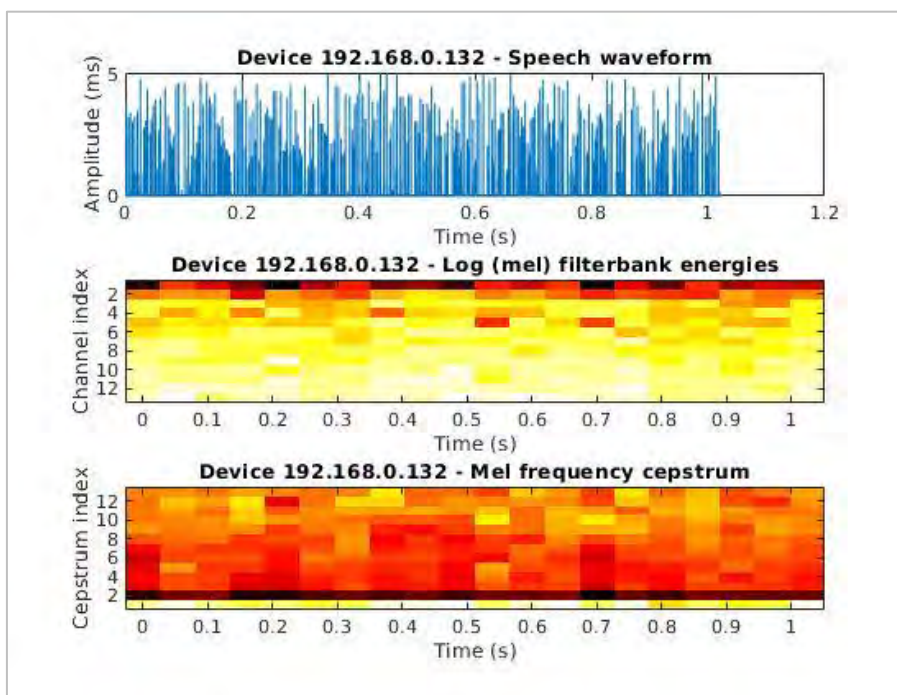


Figure B.9: Device 10 Time and MelPlots.

### Device 11: 192.168.0.136

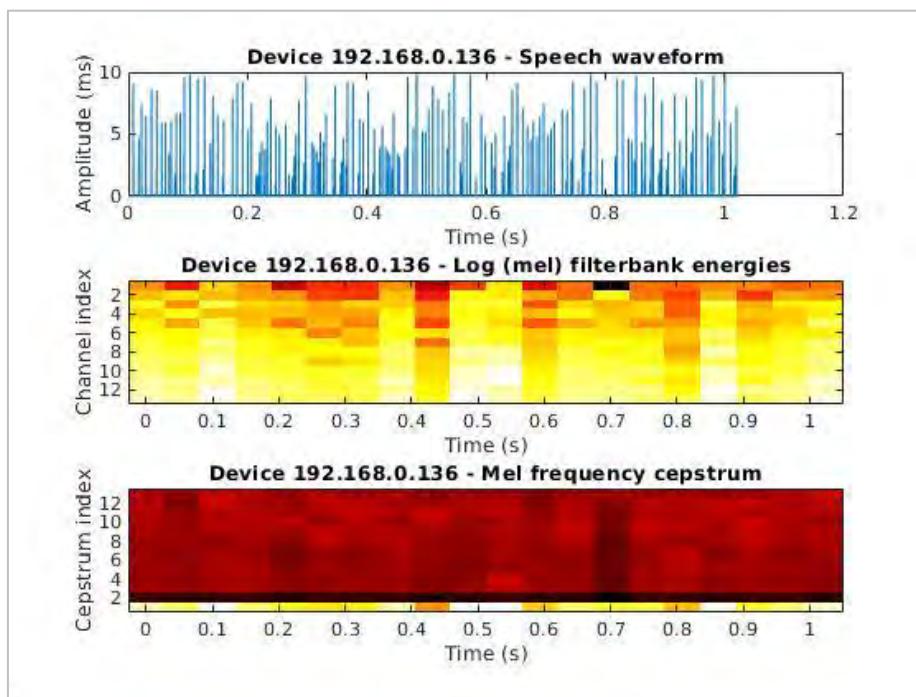


Figure B.10: Device 11 Time and MelPlots.

### Device 12: 192.168.0.140

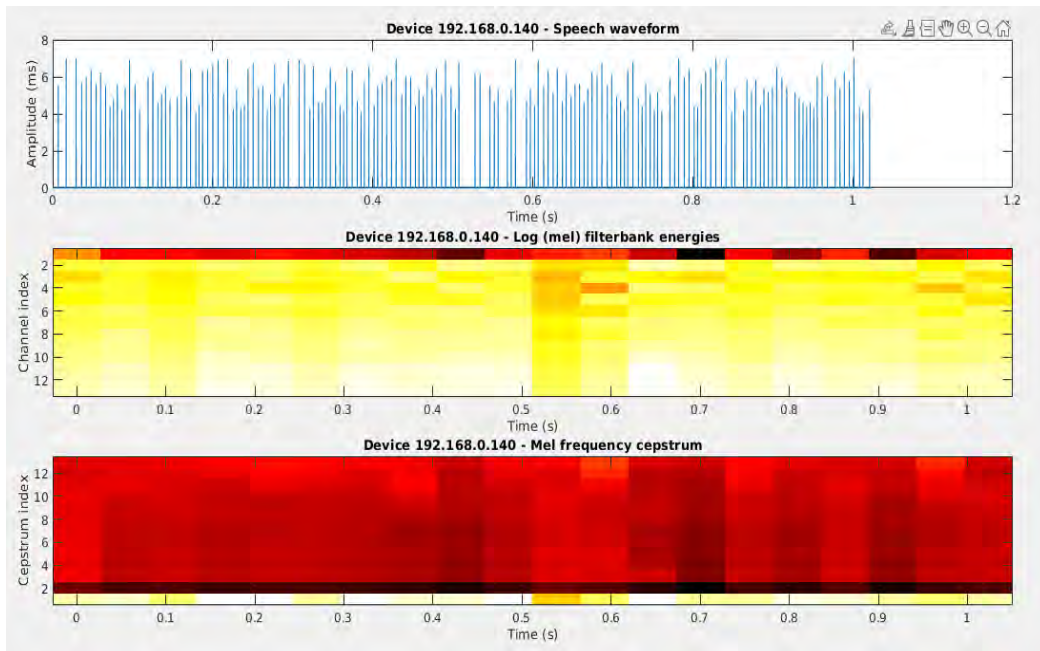


Figure B.11: Device 12 Time and MelPlots.

### Device 13: 192.168.0.141

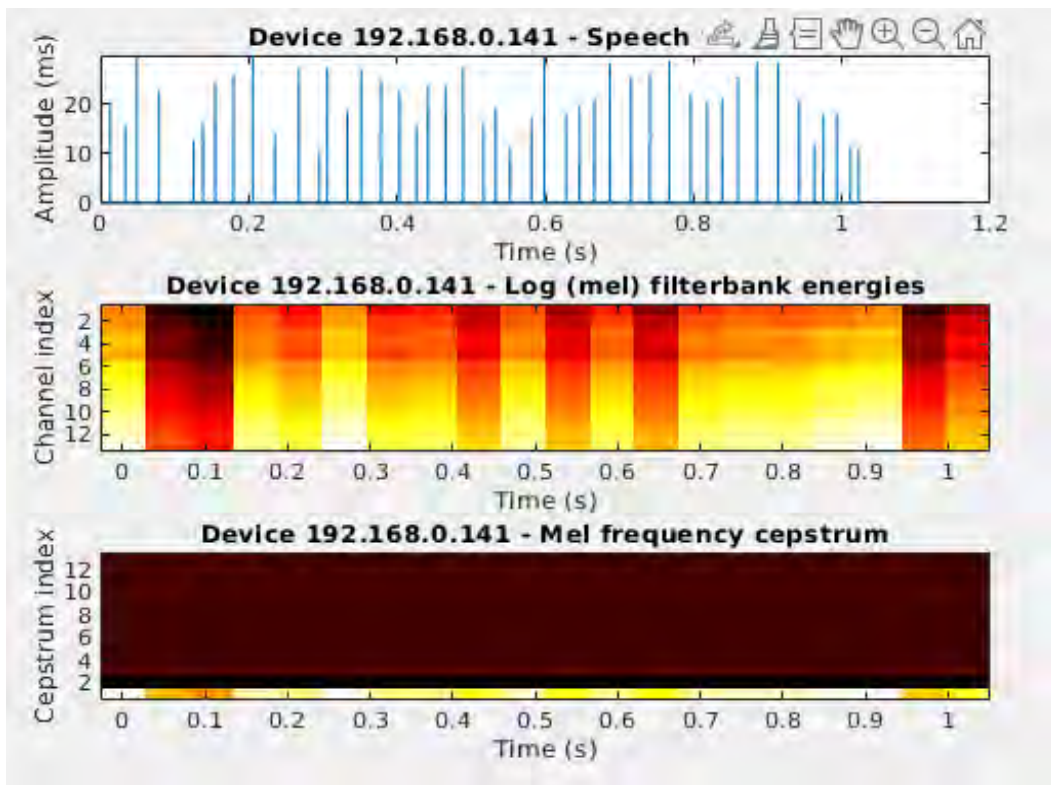


Figure B.12: Device 13 Time and MelPlots.

### Device 14: 192.168.0.142

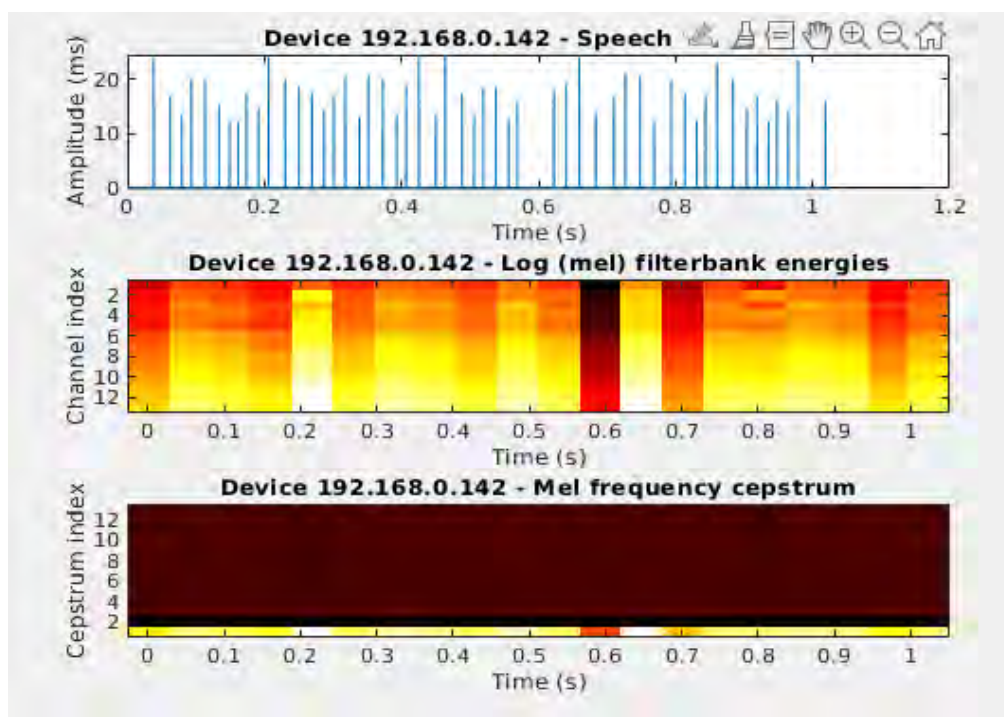


Figure B.13: Device 14 Time and Mel Plots.

### Device 15: 192.168.0.145

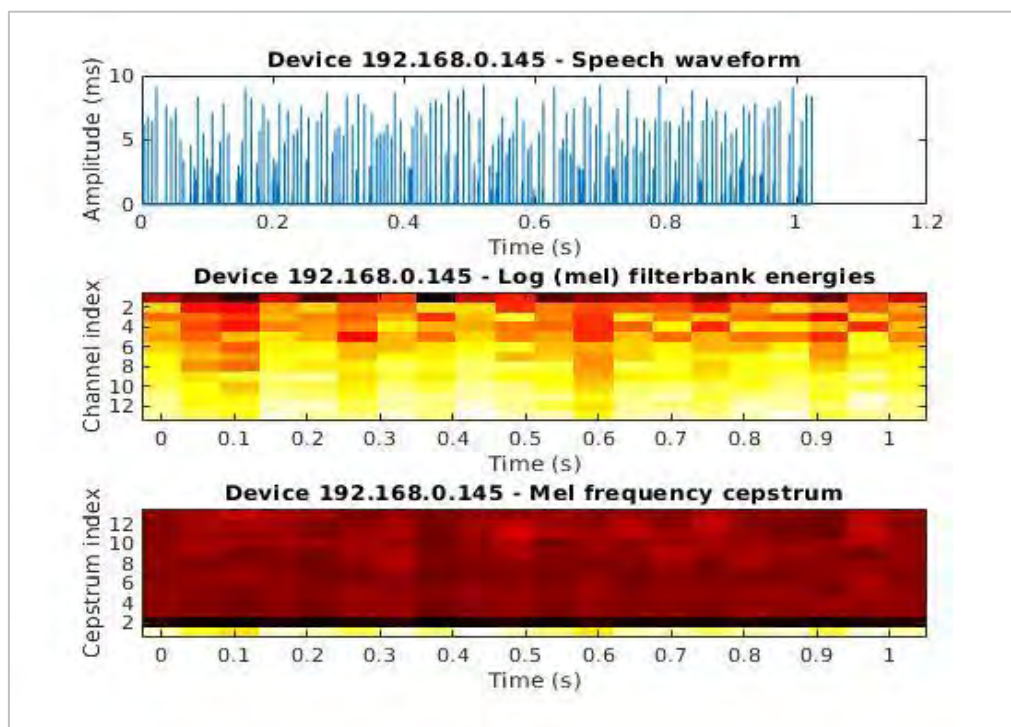


Figure B.14: Device 15 Time and Mel Plots.

### Device 16: 192.168.0.149

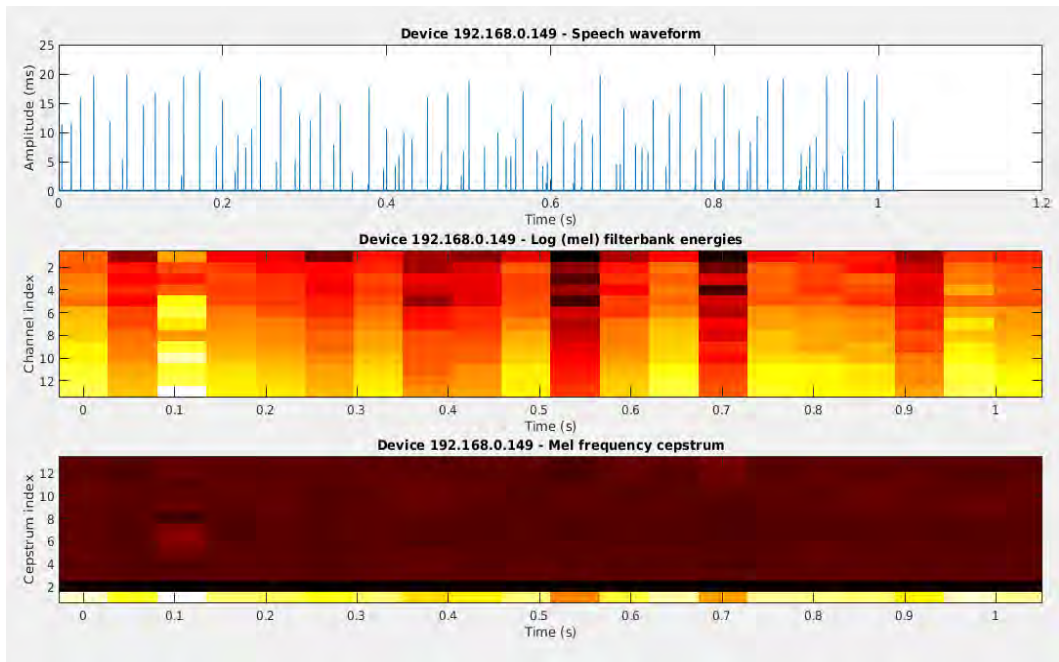


Figure B.15: Device 16 Time and Mel Plots.

### Device 17: 192.168.0.150

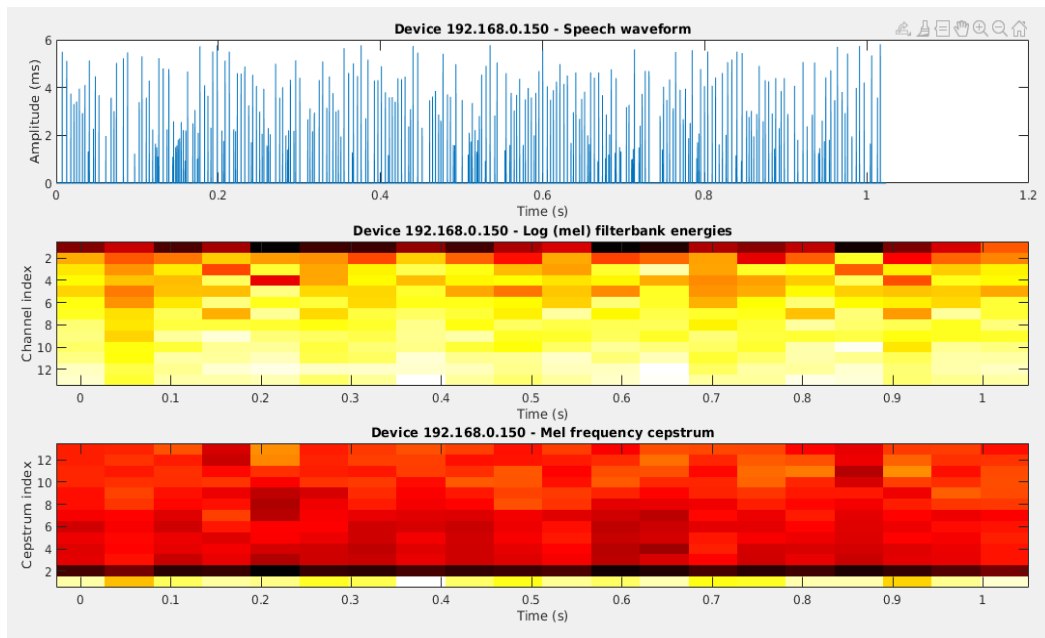


Figure B.16: Device 17 Time and Mel Plots.

## Device 18: 192.168.0.151

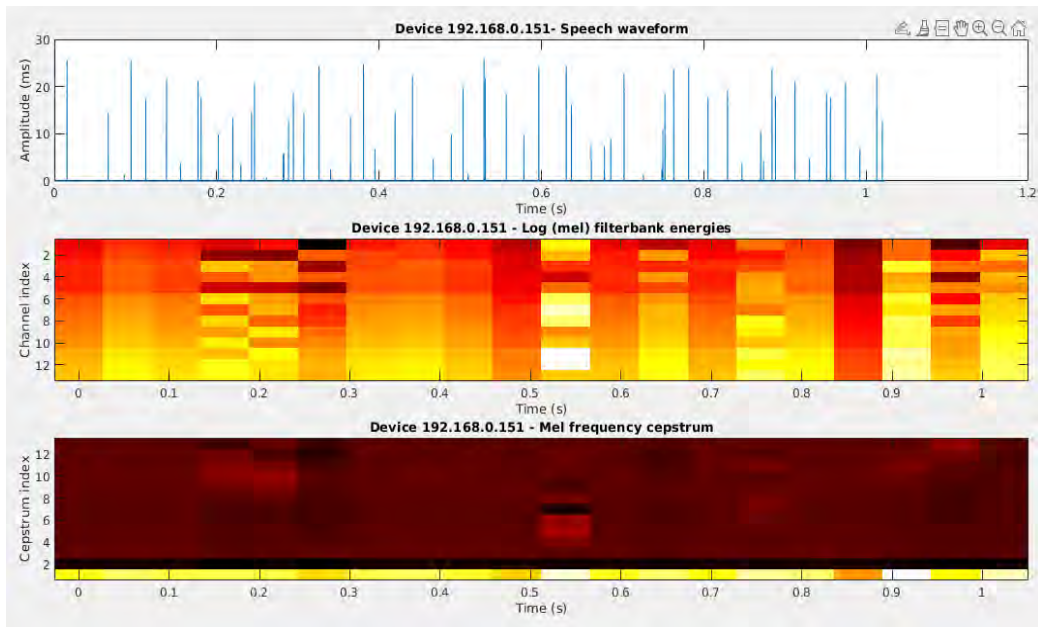


Figure B.17: Device 18 Time and Mel Plots.

## Appendix C

This appendix contains all the K-Fold confusion Matrix presented in Chapter 7.

**Fold 1, accuracy: 0.9994**

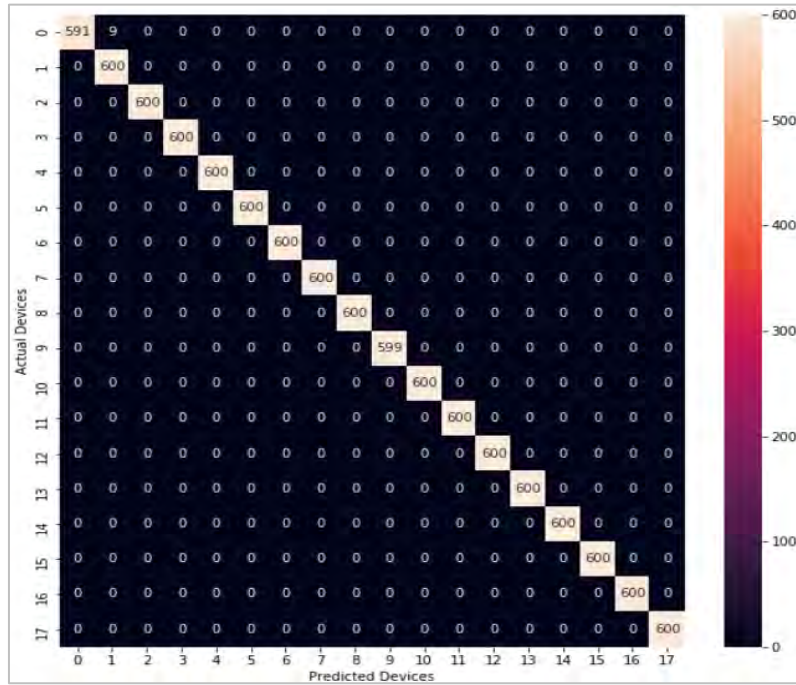


Figure C.1: Fold 1 Confusion Matrix.

**Fold 2, accuracy: 0.9993**

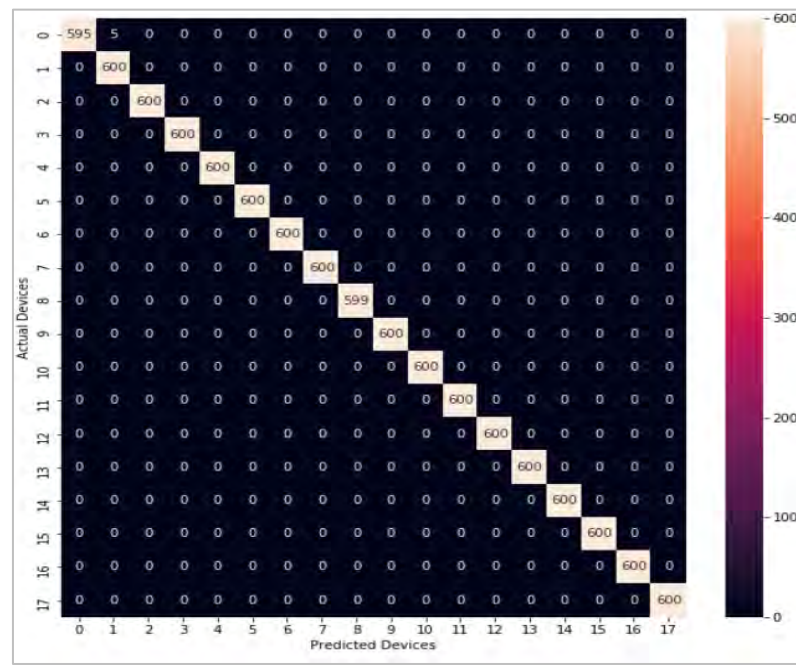


Figure C.2: Fold 2 Confusion Matrix.

**Fold 4, accuracy: 0.9995**

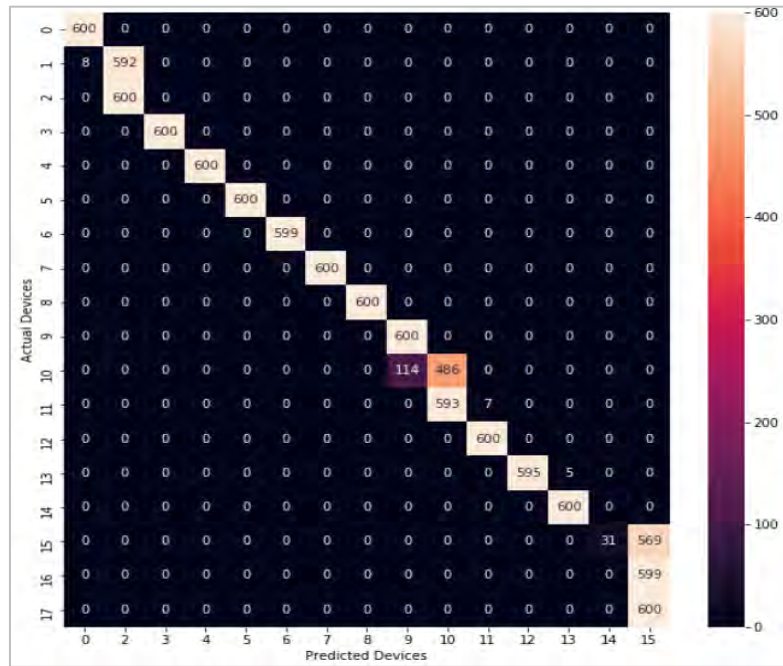


Figure C.3: Fold 4 Confusion Matrix.

**Fold 5, accuracy: 0.9994**

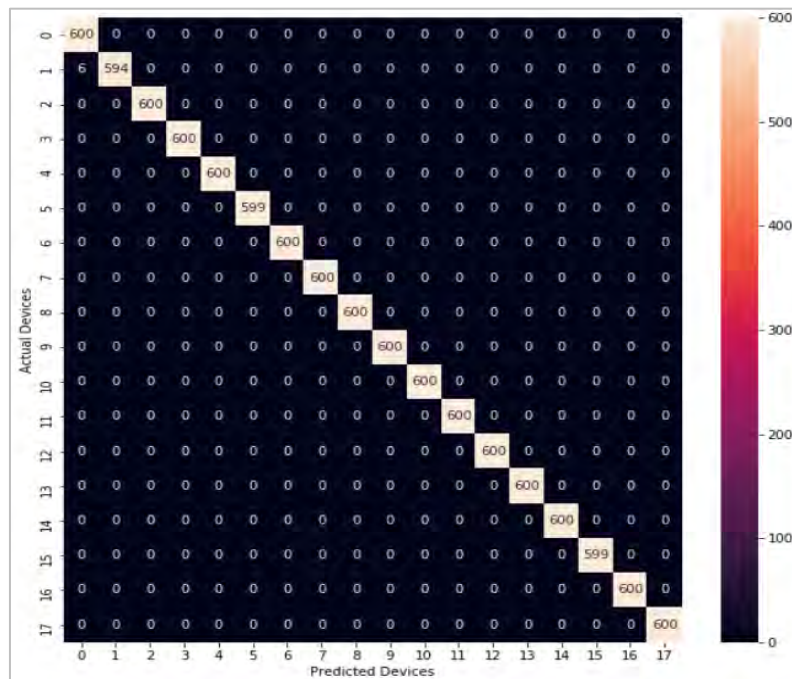


Figure C.4: Fold 5 Confusion Matrix.

**Fold 6, accuracy: 0.9990**

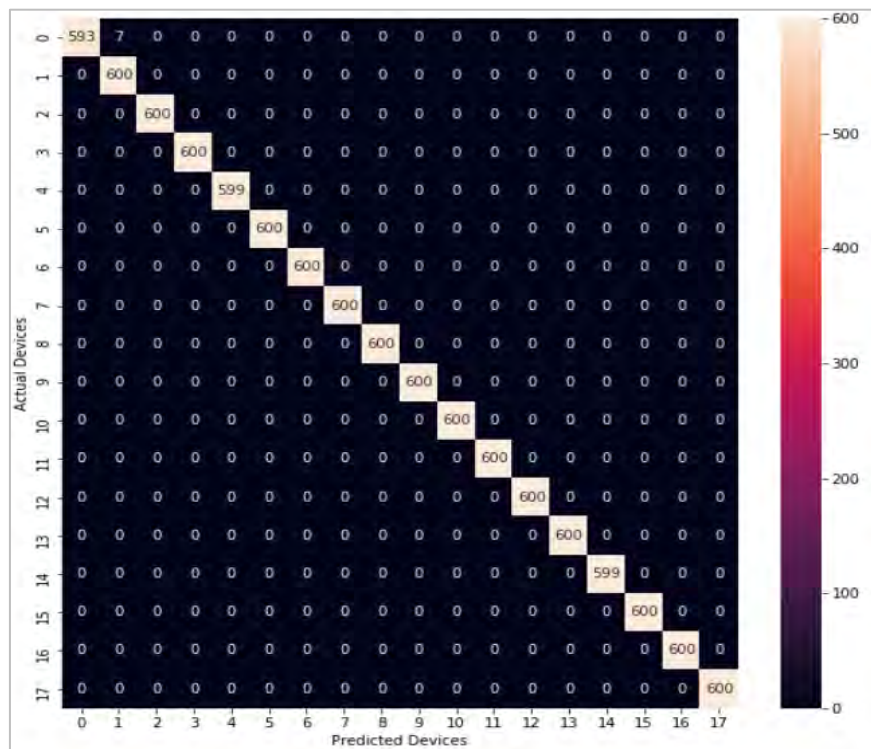


Figure C.5: Fold 6 Confusion Matrix.

**Fold 7, accuracy: 0.9994**

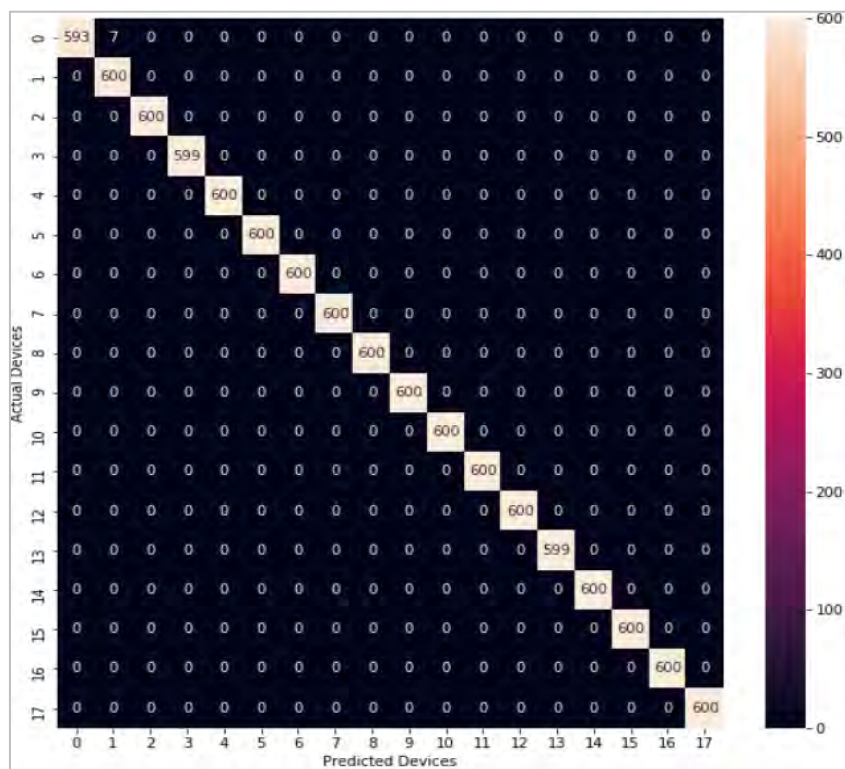


Figure C.6: Fold 7 Confusion Matrix.

**Fold 8, accuracy: 0.9995**

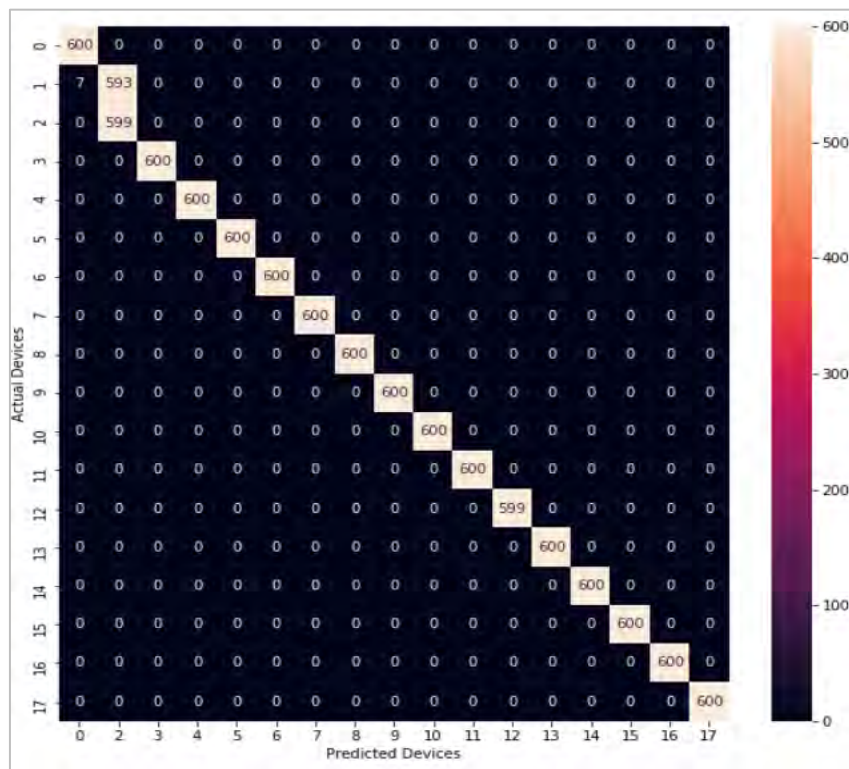


Figure C.7: Fold 8 Confusion Matrix.

**Fold 9, accuracy: 0.9993:**

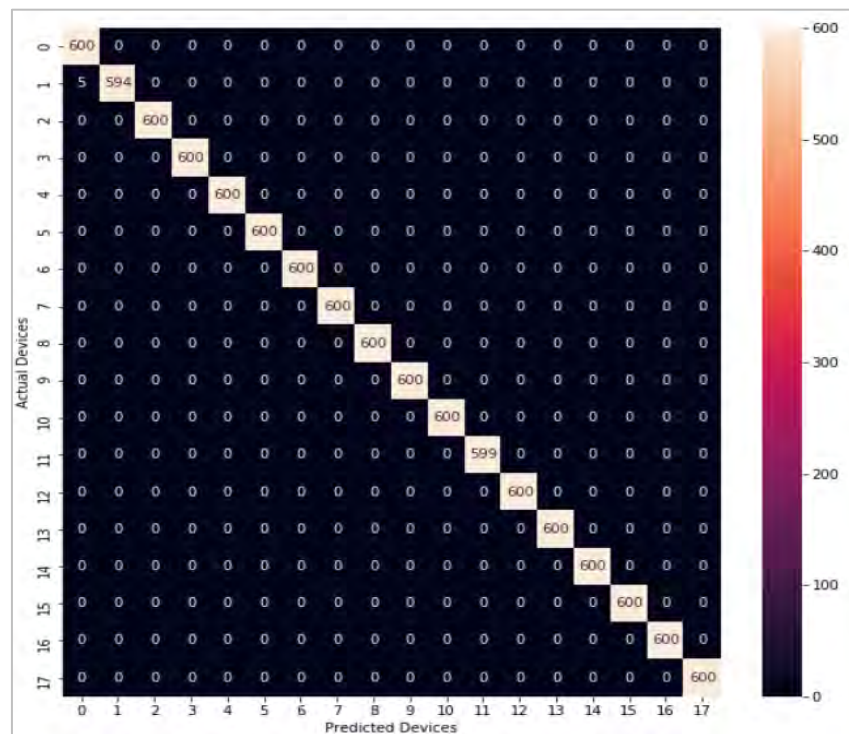


Figure C.8: Fold 9 Confusion Matrix.

Fold 10, accuracy: 0.9994

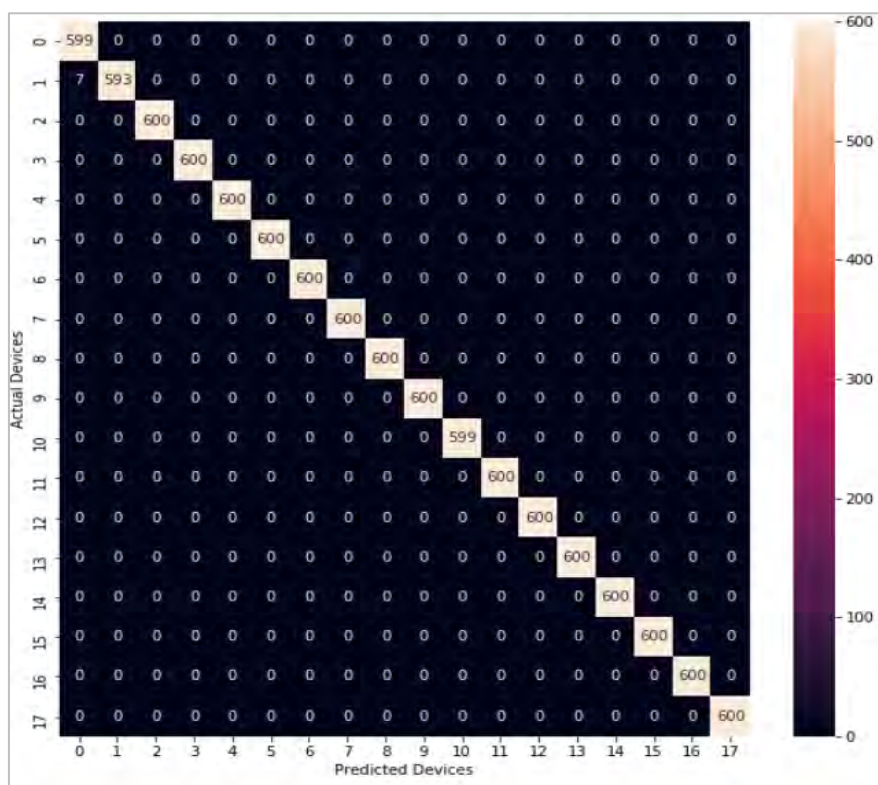


Figure C.9: Fold 10 Confusion Matrix.

## Appendix D

This appendix contains the Feature Map plots of Convolutional Layer 2 referred to in Chapter 7.

### Feature Maps:

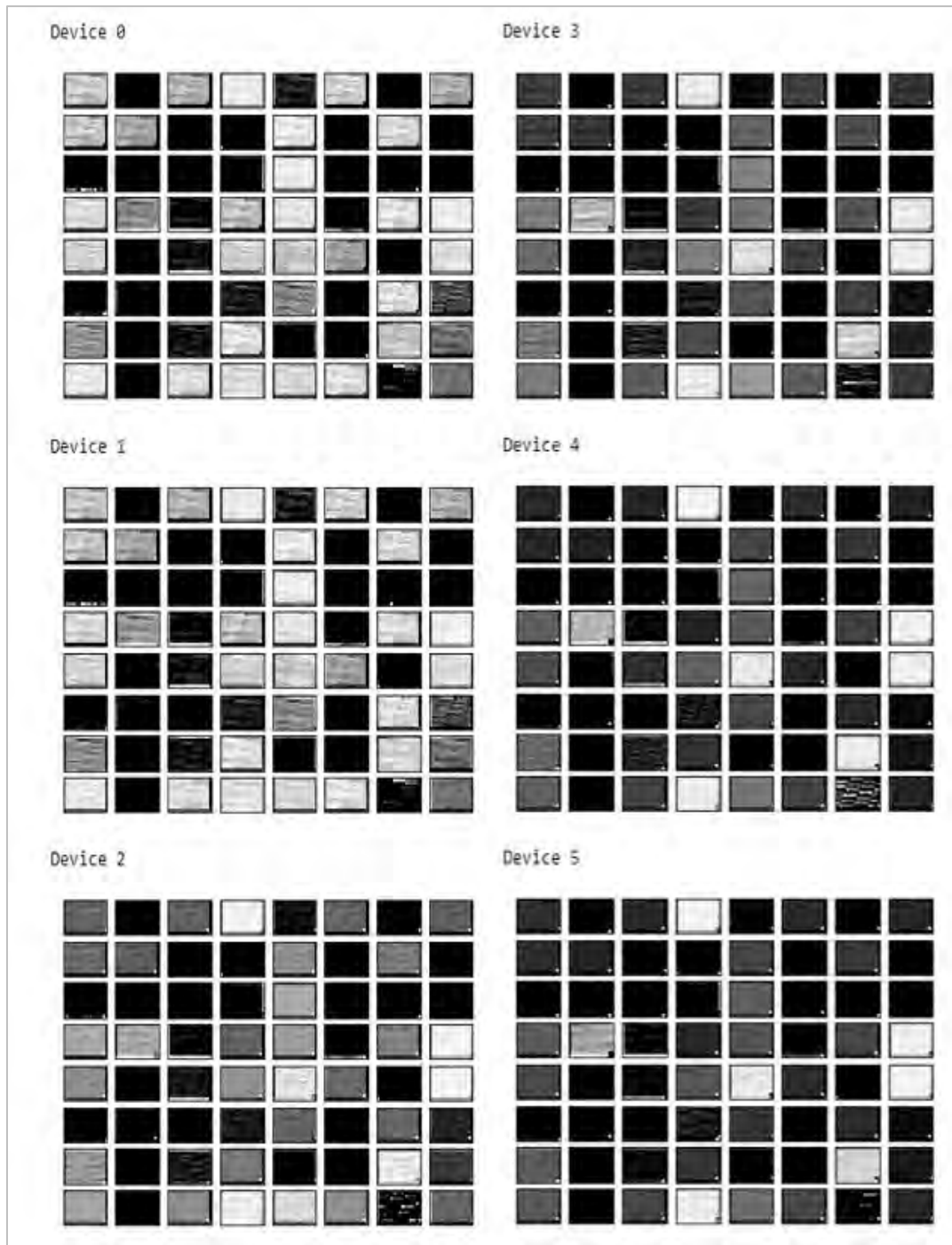


Figure D.1: Feature Map of Layer 2 Filter on Device 0 -5.

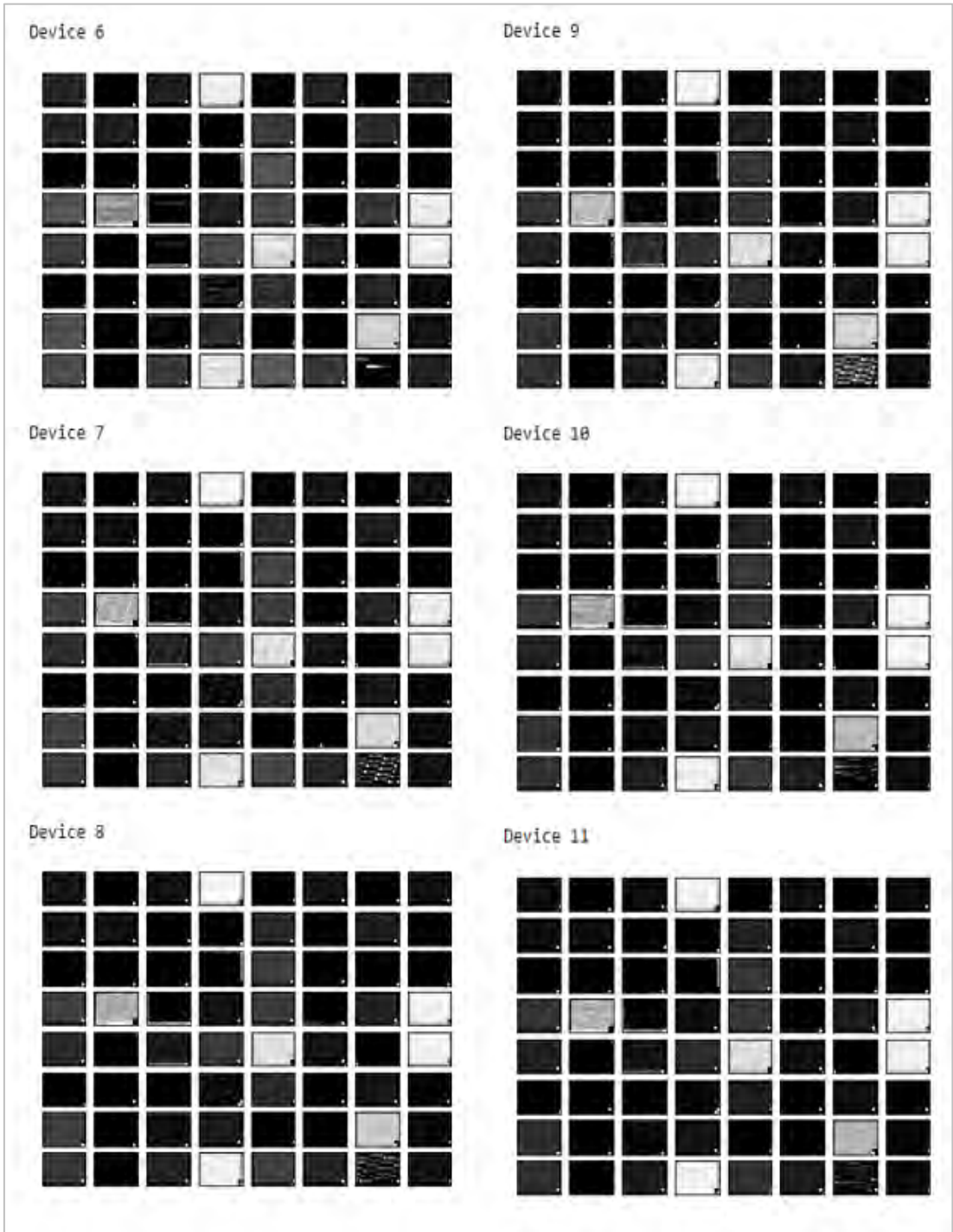


Figure D.2: Feature Map of Layer 2 Filter on Device 6 – 11.

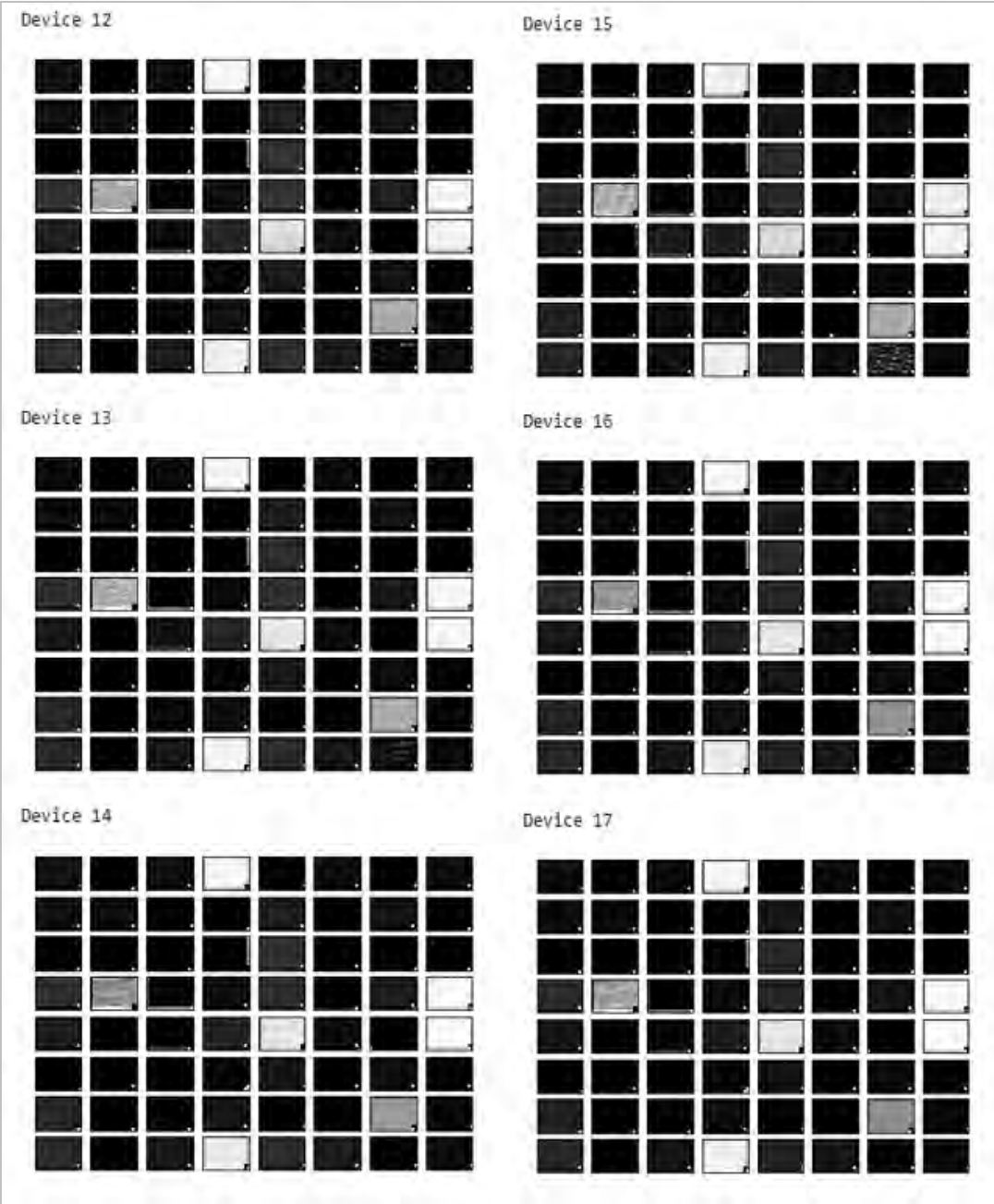


Figure D.3: FeatureMap of Layer2 Filter on Device 12 – 17.

## **Vita**

Areej Osama Mohammad was born in 1996, in Dubai, United Arab Emirates. She received her primary and secondary education in Dubai, UAE. She received her B.Sc. degree in Computer Engineering from the University of Sharjah in 2017. Since 2017, she has been working as a Client Technical Specialist in IBM Middle East.

In September 2017, she joined the Computer Engineering master's program in the American University of Sharjah. During her master's study, she co-authored one paper which was presented in international conferences. Her research interests are in Cloud Computing and Internet of Things.