UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

CLASSIFYING WI-FI FROM RAW POWER MEASUREMENTS USING A NEURAL NETWORK ADAPTED FROM WAVENET

A THESIS

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

MASTER OF SCIENCE

By

ROBERT ESTES

Norman, Oklahoma

2021

CLASSIFYING WI-FI FROM RAW POWER MEASUREMENTS USING A NEURAL NETWORK ADAPTED FROM WAVENET

A THESIS APPROVED FOR THE

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

BY THE COMMITTEE CONSISTING OF

Dr. Hazem H. Refai, Chair

Dr. Samuel Cheng

Dr. Choon Yik Tang

Dedication

To my loving parents,

Shirley and Bob,

who have been there for me whenever I needed them,

to my cherished siblings,

Amber and Terry,

whom I love dearly,

to my amazing friends,

Amy, Crys, Jessica, Kamber, Kandice, Melissa, and Tarea

who are like family to me,

&

to my dear-departed friend,

Will,

whom I miss every day.

Acknowledgements

I wish to thank Derrick Willis, Samuel Chan—fellow students at the University of Oklahoma—Alex Fortney, and Omar Aly—performed summer internships with Dr. Refai—for their help with collecting the data for this study. My thanks and gratitude go out to Michelle Farabough whom reviewed my thesis and gave much appreciated feedback. I would also like to thank Dr. Samuel Cheng and Dr. Choon Yik Tang for their review of my thesis and their part on my committee. In addition, I would also like to show my gratitude to my mentor and thesis advisor Dr. Hazem Refai for his continuous help and support throughout the last few years and to my family and friends for their love and support.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ANN	Artificial Neural Network
ARFIMA	Autoregressive Fractionally Integrated Moving Average
AUC	Area Under the ROC Curve
AWGN	Additive White Gaussian Noise
CIFAR	Canadian Institute for Advanced Research
CMU-MOSI	The Multimodal Corpus of Sentiment Intensity
CNN	Convolutional Neural Network
IoT	Internet of Things
ISM	Industrial, Scientific and Medical
IQ	In-Phase and Quadrature Components
JSB	Johann Sebastian Bach
KNN	K-Nearest Neighbors
LSTM	Long Short-Term Memory
LTE	Long Term Evolution
MNIST	Modified National Institute of Standards and Technology
OS	Operating System
P-MNIST	Permuted MNIST (See MNIST)
PTB	Penn Treebank
RF	Radio Frequency
RFID	Radio-Frequency Identification
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
SGD	Stochastic Gradient Descent
SNR	Signal-to-Noise Ratio
SVM	Support Vector Machine
TTS	Text-to-Speech
VST	Vector Signal Transceiver

Abstract

Research to identify coexisting wireless devices is becoming increasingly important, in part due to the yearly increase in internet users and wireless devices. Various machine learning algorithms, including neural networks, have been proposed and utilize a wide variety of data and feature extraction methods. Most leverage features from frequency domain because, although limited, these solutions tend to be less complex.

In this thesis, a neural network that utilizes dilated convolutions is proposed to classify Wi-Fi standards using raw power measurements. The proposed model is adapted from a previous model, namely WaveNet [1], which was used for generating raw audio in text-to-speech (TTS) applications. With this method, synthesized audio sounds more natural than other state-of-the-art TTS methods. By utilizing dilated convolutions, WaveNet has a larger receptive field with few layers that can model long-range temporal dependencies. This serves as an advantage that both recurrent neural networks (RNN) and long short-term memory (LSTM) networks do not share. Wi-Fi power measurements are collected across 802.11n, 802.11ac, and 802.11ax wireless technologies both individually and with multiple technologies coexisting across a range of various throughputs. These are used to train the proposed model (WIFINet).

Results indicate that 98.10% detection accuracy can be achieved by utilizing the proposed network. Investigations showed a convolutional neural network (CNN) with similar accuracy of 96.33%, indicating that modeling long-range temporal dependencies is not needed. At the very least, WIFINet yields little improvement over CNN for identifying wireless devices operating across the span of 802.11 technologies.

Chapter 1 Introduction

Concerns about the coexistence between wireless devices and technologies have steadily grown over the past few years, many of which utilize the Industrial, Scientific, and Medical (ISM) band. New Internet of Things (IoT) devices like personal assistants are becoming more commonplace in households. Upgrading existing wireless devices with IoT capabilities further increases the number of wireless devices. Simultaneously, the number of internet users and the total number of connected wireless devices around the world continues to grow. The Annual Internet Report from Cisco [2] projects that by 2023, an estimated 5.3 billion internet users and 29.3 billion networked devices will vie for connectivity. This figure reflects an increase of 3.9 billion users and 18.4 billion devices since 2018. Most devices are expected to leverage wireless technology utilizing either the 2.4 GHz or 5 GHZ ISM bands.

Investigating coexistence becomes more critical as the ISM band becomes more congested. Because technologies compete for limited spectrum resources, successful coexistence and resource sharing with limited interference is optimal. As such, studying ways to identify wireless devices that coexist (i.e., no interference) on shared bands and novel methods for sharing resources will be beneficial for device development.

1.1 Thesis Summary

While many methods have been proposed to identify coexisting wireless devices utilizing a range of wireless technologies, another method is suggested in this thesis for utilizing a neural network capable of modeling long-range temporal dependencies. If using an IQ sampling rate of 10 MS/s, then sampling just 1 ms would generate a sample with 10,000 timesteps. Hence, RNN and LSTM models should not be utilized. Instead, a CNN architecture is proposed. Dilated

convolutions will result in long-range temporal modeling capabilities. The model was tested for its ability to identify wireless technologies for both independent and coexisting operations covering 802.11n, 802.11ac, and 802.11ax. Additionally, the model was compared with alternative machine learning models.

1.2 Benefits of Dilated Convolution

Dilated convolutions can be used when an increase in the receptive field (i.e., number of inputs used to generate each output) is desired while maintaining a minimal number of layers. In traditional convolution, receptive fields grow linearly. A new convolution layer must be added for each additional output. In dilated convolution, a spacing—most commonly called a dilation rate—is introduced between kernels, effectively widening the receptive field without adding additional layers. Stacking dilated convolutions with ever increasing dilation rates can achieve an exponentially higher receptive field with fewer layers when compared to non-dilated convolutions. For example, a dilation depth of three results in a receptive field of eight, while a non-dilated convolution with three layers results in a receptive field of only four, as shown in Figure 1-1. Given that the dilation rate is increased by a power of two for each additional dilated layer, layer depths ranging from [3 4 5 6 7 8 9 and 10] results in receptive field increases of [8 16 32 64 128 256 512 and 1,024], respectfully. If non-dilated convolution was used for a depth of 10 layers, the receptive field would be 11 inputs wide. To obtain a similar receptive field of 1,024, a non-dilated convolution network must be 1,023 layers. With dilated convolution, outputs can be generated to model long-range temporal dependencies since a large receptive field is possible with few layers. [1], [3], [4], [5], [6], [7].

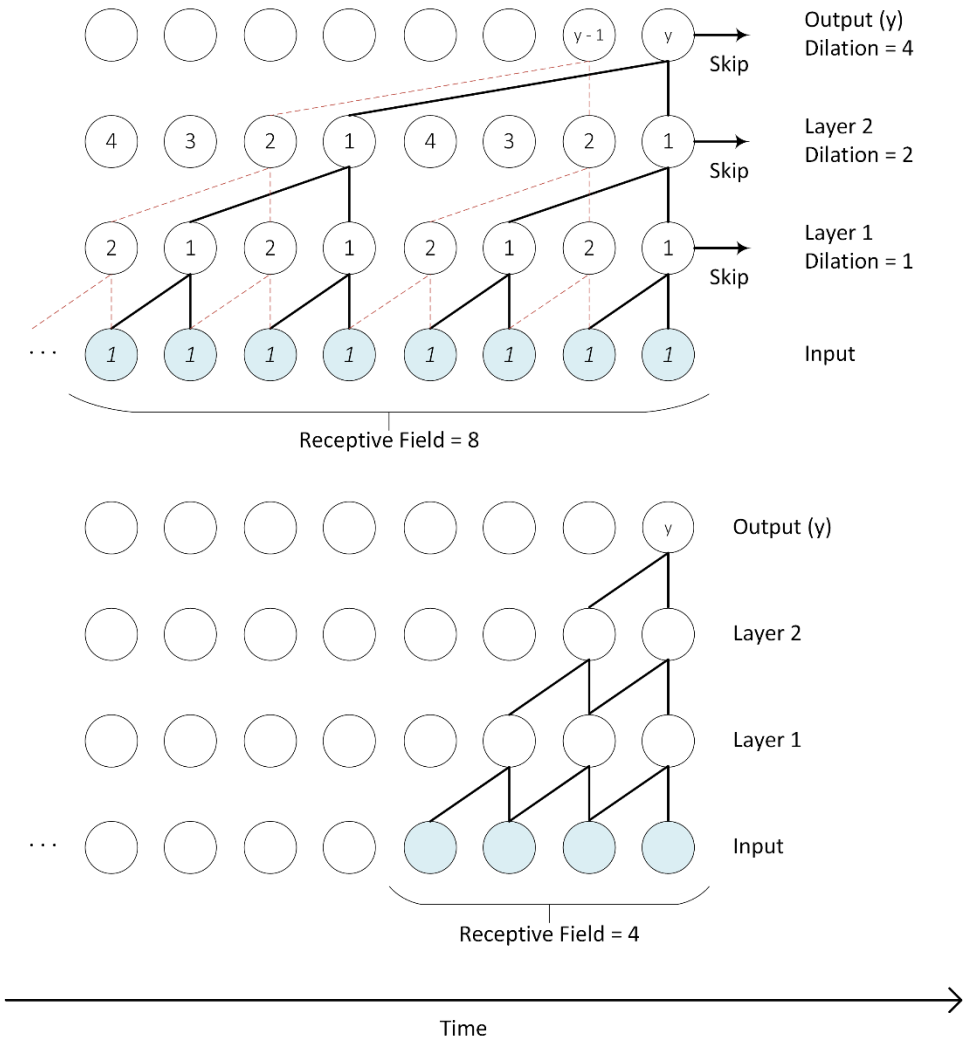

Figure 1-1. Dilated Convolution (top) versus Non-Dilated Convolution (bottom).

Chapter 2 Related Work

2.1 RNN and LSTM Models for Time Series Datasets

RNN or LSTM architectures are commonly used in conjunction with time series data. However, both architectures suffer from vanishing gradients when sequence lengths are long (e.g., approximately 50 timesteps for RNN and 500 timesteps for LSTM).

Authors in [8] provide a statistical means of testing whether RNN and LSTM networks have long memory. Their conclusion proved that they do not. They also proposed a new definition for long memory networks which required weights to decay at a polynomial rate. Testing involved minimally modifying RNN and LSTM models to use a polynomial decay rate and training the model on various datasets, including a generated dataset using autoregressive fractionally integrated moving average (ARFIMA) of sequence lengths 4,001; the Dow Jones Industrial Average with sequence lengths of 5,030; hourly traffic volume for an interstate with sequence lengths 1,860; and tree ring measurements with sequence lengths of 4,351. Additionally, two sentiment analysis datasets, CMU-MOSI and a paper review set, were used for classification. While their modified RNN and LSTM models showed improvement over the un-modified version, performance was poor for sentiment analysis datasets with accuracies of approximately 31% and 40%, respectively.

Authors in [9] showed that CNNs could outperform RNNs and LSTMs in many situations. Their datasets included the adding problem—a common stress test used for sequence problems—with sequence lengths of 600; the MNIST and P-MNIST datasets with sequence lengths of 784; a copy memory test—prior works with sequence models used this— with sequence lengths of 1,000; the JSB Chorales and Nottingham music dataset with sequence

lengths of 382; and several datasets for word and character modeling with vocabulary sizes ranging from 10,000 to 268,000. CNN drastically outperformed RNN in all cases and outperformed LSTM in all cases except word modeling. Researchers pointed out that in the latter case, PTB dataset is smaller; on the much larger Wiki-103 dataset, CNN outperforms LSTM by achieving a much lower perplexity.

Authors in [10] proposed a modification to an RNN / LSTM network that allows modeling longer term dependencies. They proposed "… adding an unsupervised auxiliary loss to the original objective …," which "… forces RNNs to either reconstruct previous events or predict next events in a sequence, making truncated backpropagation feasible for long sequences" [10, p. 1]. Researchers evaluated this method on datasets with various sequence lengths, including MNIST and P-MNIST with lengths of 784; CIFAR10 with lengths of 1,024; StanfordDogs with lengths ranging from 1,600 to 16,000; and DBpedia with lengths of 300. The proposed method improved the model to better generalize longer sequences. When looking at results of datasets with long sequence lengths, the researchers achieved accuracies of 65% to 72% on CIFAR10 compared to un-modified versions achieving 49% to 59% and accuracies of approximately 12% on the StanfordDogs dataset with length of 2,000 which decreased to below 6% as length increased to 16,000. Authors noted that "… pursuing useful accuracy with non-convolutional models is not our main goal" [10, p. 5]. Results indicated that without more extreme modifications or tuning, even methods proposed to increase the network's ability to model longer term dependencies still demonstrate disappointing results when sequence lengths rise above 1,000.

2.2 Identification of Wireless Technologies using Neural Nets

Authors in [11] used a CNN to identify coexisting wireless devices using features from the frequency domain. Their network identified individual and coexisting wireless technologies present in an environment and encompassed 802.11n, Bluetooth, and Zigbee. Raw power measurements were recorded across the 80 MHz spectrum at various SNRs and used to train their model, which achieved an accuracy of 93% with the highest SNR. Accuracy decreased to 14% for the lowest SNR of 0 dB, even though it performed decently well for SNRs at or above 10 dB. Provided figures showed that 802.11n, Wi-Fi, Bluetooth, and Zigbee have unique and distinguishable spectrograms that prove useful for identification.

Authors in [12] used an expanded approach, utilizing a CNN to classify 802.11ac, 802.11ax, 802.11a, 802.11n, and Long-Term Evolution (LTE) cellular using the spectrograms. In addition, researchers demonstrated that by utilizing a denoising autoencoder prior to CNN, high accuracy classification in non-ideal channels is achievable by using lower-dimensional and denoised representations of the spectrograms. The group tested their proposed model against five noise scenarios for 802.11 Wi-Fi—three delay spread scenarios with and without pathloss, one doppler scenario, and one AWGN scenario—and against four noise scenarios for LTE cellular—two LTE moving with and without fading, one LTE fading, and one AWGN—at SNR values of 20 dB, 15 dB, 10 dB, 5 dB, and 0 dB. Maximum achieved accuracy across the five scenarios was 100% with an average accuracy of 91% across all tests and 95.44% across tests with SNR level between 10 dB and 20 dB. The lowest accuracy achieved was 55% in the LTE AWGN scenario with SNR level of 0 dB.

Authors in [13] used network traffic flows to classify new and unseen IoT devices. Since traffic flows varied in length depending on the device, the researchers segmented the traffic into

sub-traffic flows of a fixed five-minute time interval. They extracted features related to number of packets (e.g., total, received, transmitted), packet statistics (e.g., max, min, mean), and protocol-related features. A cascaded neural network containing LSTM layers followed by CNN layers was used. Devices ranged from smart speakers, smart cameras, baby monitors, and printers, among others. Researchers classified devices as hubs, electronics, cameras, and switches and triggers. Devices were chosen from each category; some were purposely left out to test the network's ability to classify previously unseen devices into one of the specified categories. Accuracy was 74.8%, which outperformed other tested models. The LSTM network achieved 65.4% accuracy; SVM was 58.5% accurate; and CNN was 56.3% accurate. The work presented in this thesis is similar, as segmented sub-traffic flows are very similar to time series data. The proposed model [13] outperformed standalone LSTM and CNN networks, indicating that cascading networks can improve performance. However, results indicated that to achieve higher accuracies, a different approach might be necessary.

Authors in [14], proposed using an LSTM to classify signals using time domain amplitude and phase information. Researchers indicated that state-of-the-art "… results on high SNRs (0 to 20dB) are achieved without using complex CNN-LSTM models" [14, p. 444]. Their proposed model achieved accuracies of 90% for high SNRs. Authors noted that using IQ samples instead of amplitude and phase information results in extremely poor performance. Findings from this thesis demonstrated similar poor performance (See Section 3.2). However, the proposed method in this thesis utilizes only the I-component of the IQ samples, which demonstrated high accuracy performance.

2.3 WaveNet

Authors in [1] introduced the WaveNet architecture for generating raw audio waveforms, which served as the basis for the proposed model in this thesis. Researchers used causal dilated convolutions, permitting a larger receptive field that grows exponentially, as discussed previously in Chapter 1. The receptive field enables the network to model long-range temporal dependencies. The model is fully probabilistic with probability distribution outputs for each timestep; it also uses newly generated samples at each timestep to predict future samples. Raw audio is typically 16-bit, which requires distributions of 65,536 values at each time step (e.g., 16 kHz would result in 16,000 distributions every second). To reduce the dimensionality of the probability distributions, raw audio signals were quantized using μ-law quantization, which represents the raw audio signal as 8-bit quantized versions and reduces distributions to 256 values. In addition, both skip and residual connections were used in the stacked residual blocks, which contained dilated convolutions and gated activations. Output was fed into the next block to prevent vanishing gradients and to reduce convergence time. Researchers demonstrated that the model performed extremely well when applied to music audio modeling and speech recognition. The model could also outperform then-current top TTS systems when considering output authenticity. Currently, Google uses an updated version of WaveNet to power Google Assistant voices for US English and Japanese translations. [3], [4], [5], [6], [7]

2.4 WaveNet Adapted for Artist Classification from Audio Samples

Authors in [15] demonstrated that WaveNet architecture can be adapted to work as a classifier, adapting it to classify artists based on audio samples. The architecture proposed herein encoded 16 kHz audio samples to a feature space of (16,000, 40) by utilizing casual and dilated

convolutions, much like WaveNet. The encoded audio sample was fed into a series convolution and pooling layers for down-sampling, and then into a softmax layer for classifying one of 20 artists. The model [15] achieved an F-score of 0.854, which researchers confirmed is better than then-current state-of-the-art methods given the same experimental conditions.

Chapter 3 Classifying Wi-Fi from Raw Power Measurements

A neural network was used to determine whether devices in a given environment utilized one or more Wi-Fi standards by utilizing raw power measurements. The neural network was inspired and adapted from WaveNet architecture [1]. It was hypothesized that the architecture would excel at this type of classification due to the time-varying nature of power measurements, size of samples, and ability of WaveNet to consider many thousand timesteps for modeling the long-range temporal dependencies with few layers by utilizing dilated convolution. Data was collected across many experiments with independent and coexisting wireless technologies across the 802.11n, 802.11ac, and 802.11ax standards. Resulting data was used to train and test the proposed model's ability to classify wireless technologies active in the environment at the time power measurements were documented.

3.1 Data Collection

Experiments were conducted in Building 5 Blockhouse at the University of Oklahoma campus in Tulsa, OK over a period of months in 2020 summer and fall semesters and 2021 spring semester.

3.1.1 Equipment

Four MikroTik 953GS-5HnT-RP RouterBOARDs (See Figure 3-1) compliant with 802.11n and 802.11ac were used as both the access point (Tx) and station (Rx) for 802.11n and 802.11ac networks. One Asus RT-AX88U, 802.11ax compliant router (See Figure 3-2) served as the access point (Tx) for the 802.11ax network. One Dell Precision 5540 laptop running Windows 10 equipped with iPerf3 software for measuring network performance was installed

and used as the server for the software and to access the ethernet-connected Asus router. One Lenovo ThinkPad T470s laptop running Windows 10 acted as the station (Rx) for the 802.11ax network. One NI PXIe-5644R Vector Signal Transceiver (VST) (See Figure 3-3) running Windows 7 and with proprietary LabVIEW software [16] installed was utilized for acquiring raw power measurements. Finally, one Toshiba R835-P56X running Windows 10 was utilized as the host computer for the four MikroTik RouterBOARDs and to remotely connect to the VST; it was connected to each component using one Netgear Gigabit Switch GS116 via ethernet.

Figure 3-1. MikroTik RouterBOARD.

Figure 3-2. Asus router.

Figure 3-3. NI vector signal transceiver.

3.1.2 Experimental Setup

Three Wi-Fi networks were setup and configured to utilize the 802.11n, 802.11ac, and 802.11ax wireless-networking standards. Each network consisted of an access point (Tx) and station (Rx). The 802.11n and 802.11ac networks utilized the four MikroTik RouterBOARDs as

both access points and stations were setup on two tables separated by 6 feet. Notably, the networks were separated by 2.5 feet. The 802.11 ax network was composed of an access point, the Asus Router, station, and Lenovo ThinkPad and placed on two additional tables separated by 12 feet. This network was separated 2.5 feet from the nearest MikroTik RouterBOARD. The VST and Dell Precision laptop shared a table with the Asus Router, and the Toshiba host laptop shared a table with the Lenovo ThinkPad. All table surfaces were approximately 2.5 feet high. The experimental setup and devices used in this setup are depicted below in more detail (See Figure 3-4, Figure 3-5, and Table 3-1).

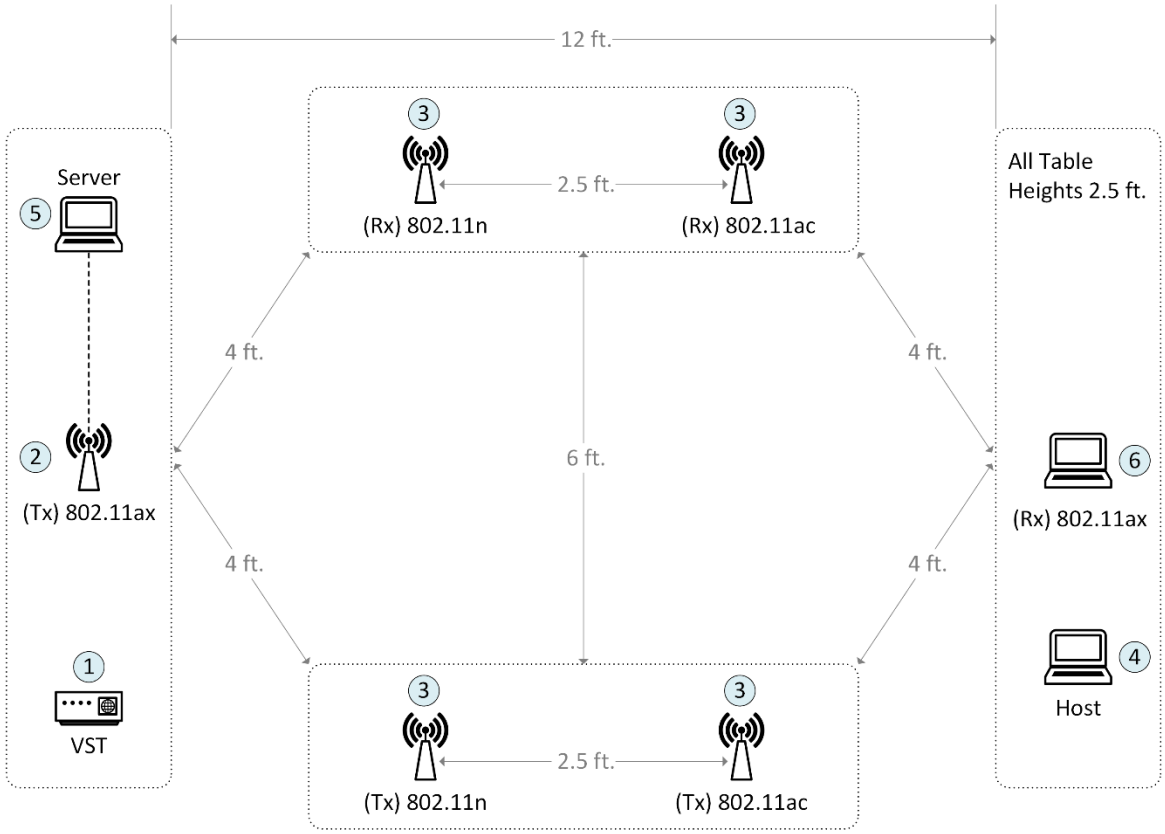

Figure 3-4. Experimental setup.

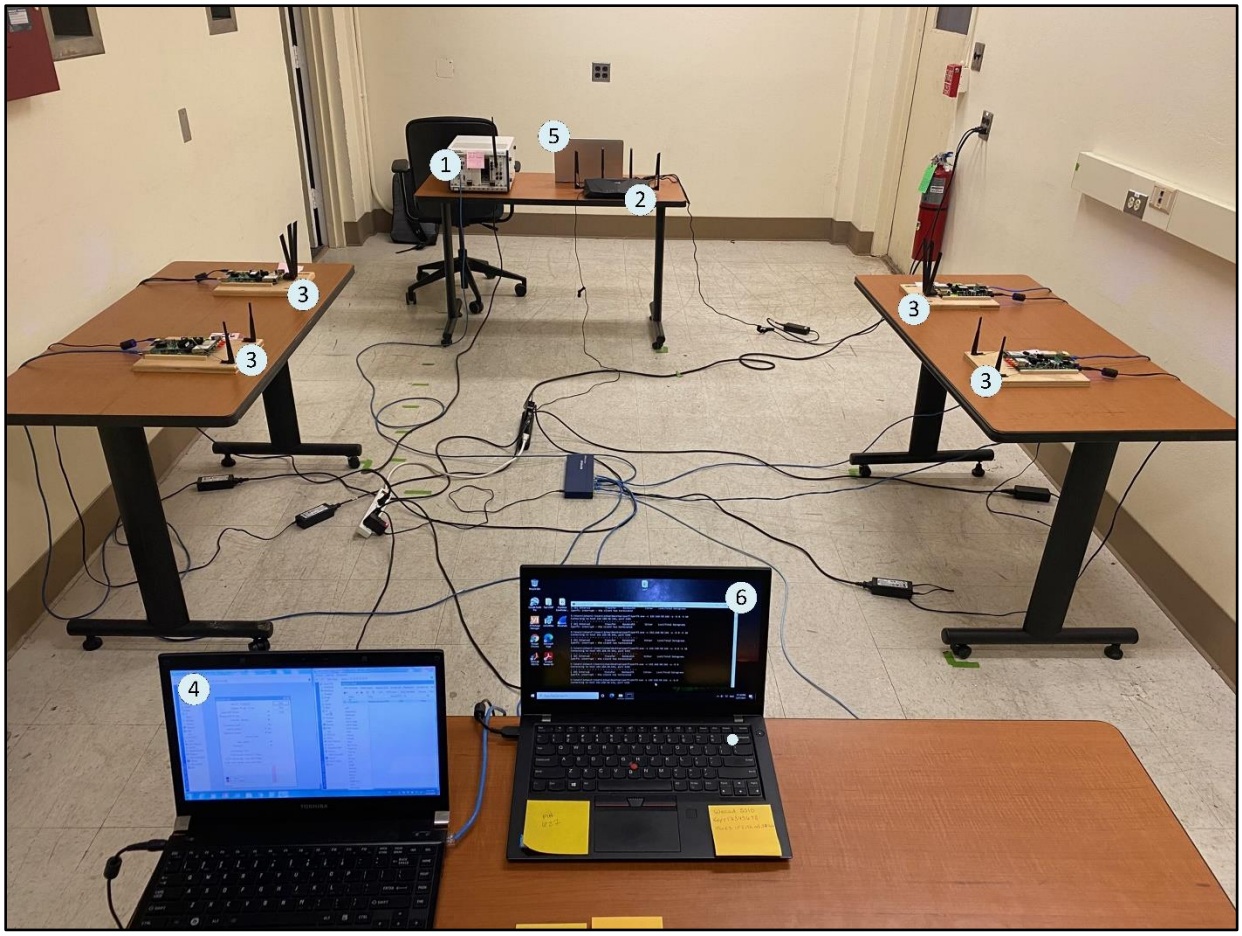

Figure 3-5. Image of experimental setup in lab.

Table 3-1. Device Details for Experimental Setup

ID	Device	Interface	OS	iPerf
1	NI PXIe-5644R	Ethernet	Windows 7	-
2	Asus RT-AX88U	WLAN	Windows 10	3.1.3
3	MikroTik RouterBOARD 953GS-5HnT-RP	WLAN	RouterOS	-
4	Toshiba R835-P56X	Ethernet	Windows 10	-
5	Dell Precision 5540	Ethernet	Windows 10	-
6	Lenovo ThinkPad T470s	WLAN	Windows 10	3.1.3

The Asus router and MikroTik boards had channel bonding enabled, which allowed for full channel utilization of 160 MHz width for the Asus router and 40 MHz width for the MikroTik boards. All were set to operate on channel 36 of the 5GHz ISM Band with a 5,180 MHz center frequency. Features were extracted from measurements taken by the VST using installed LabVIEW software [16]. Features included average duty cycle, average power, raw duty cycle, and raw power. Power values were expressed as in-phase and quadrature (IQ) components. LabVIEW software configuration is detailed below in Table 3-2.

Table 3-2. LabVIEW Software Configuration

Setting	Description	Value
IQ Rate	Sampling rate of the baseband IQ data.	10 MS/s
Gain	Aggregate gain applied to the RF signal.	0 dBm
Frame Size	IQ data frame size to be analyzed per iteration.	72,600
Run Period	Duration of the scan.	60 s
Threshold	Threshold used to calculate the duty cycle.	-59 dBm

3.1.3 Collection

Ambient noise was baselined before data collection began to ensure no interfering signals were present in the environment. Daily testing re-affirmed this, and ambient noise remained below -73 dBm. Baseline testing was conducted separately for each of the three networks to evaluate maximum throughput performance. Each baseline test was repeated five times, and the 802.11n, 802.11ac, and 802.11ax networks achieved maximum throughput of 250 Mbps, 340 Mbps, and 956 Mbps, respectively. This maximum data rate was then divided into five limits that could be used to artificially limit throughput to simulate various signal qualities (See Table 3-3). Lower throughputs were used for simulating low quality signals, and higher throughputs for simulating high quality signals.

Table 3-3. Throughput Limits for each Wi-Fi Network

Specification	Limits (Mbps)				
802.11ax	190.0	380.0	570.0	760.0	956.0
802.11ac	68.0	136.0	204.0	272.0	340.0
802.11n	50.0	100.0	150.0	200.0	250.0

Collection consisted of a series of tests in which one to three networks operated at a specified limit and each test was repeated five times. Tests were composed of individual networks with each limit and each possible limit combination for coexistence. One exception was made to test all three networks coexisting wherein limits were reduced to the combination of individual minimum, median, and maximum limits for all five. The latter test was performed to decrease the number of tests required for this particular case, which otherwise would have been 625—a number that eclipsed all other testing combined. Instead, 135 tests were conducted, which was more feasible.

3.2 Exploratory Data Analysis

A total of 585 tests were performed—25 for each of the three individual networks, 125 for each of the three cases of two coexisting networks, and 135 for a single case of three coexisting networks. Each test resulted in four binary (.bin) files for average and raw power, and average and raw duty cycle. Of these, raw power was used. Total size of 585 raw power binary files was approximately 608 GB. Each contained approximately 1.054 billion interleaved IQ components of raw power measurements in dBm, which was stored as signed 8-bit integers. Maximum value observed in the data was -3.0 dBm, and minimum value observed was -85.0 dBm. A total of seven classes was labeled with numbers ranging from zero to six (See Table

3-4). Number of classes was determined cases of individually operating networks, as well as cases of coexistence. Different limits do not constitute a new class, but rather ensure each class contains data with various signal qualities that were simulated with throughput limits. Early testing with various model architectures aided in determining that the I component alone would be enough for training the neural network. In fact, test results indicated that model accuracy would be quite low using only the Q component. Including both components achieved decent results, although results were poorer compared to training results using only the I component. A decision was made to discard the Q component during data preprocessing.

Table 3-4. Description of Classes and Assigned Labels

Label	Class Description	# Networks
0	802.11ax	1
1	802.11ac	1
2	802.11n	1
3	802.11ax / 802.11ac	2
4	802.11ax / 802.11n	2
5	802.11ac / 802.11n	2
6	802.11ax / 802.11ac / 802.11n	3

3.3 Data Preprocessing

Each of the 585 binary files were processed in the same way. File data was loaded into memory; IQ data was de-interleaved; Q-component was discarded; data beginning and end was trimmed; and then data was randomly sampled.

Labels were generated based on file name and originating directory. Files were stored in a directory that indicated the class to which they belonged. Files were named with a five-digit number. The first three digits were a zero-padded number indicating the set of tests the file

belonged to. The last two digits were a zero-padded number indicating test number in the test set (See Figure 3-6).

After all files had been processed and labels generated, data was stored in two arrays and saved to disk using the binary NumPy format (.npy). Data was saved in single-precision floating-point format (float32) and kept in dBm to allow testing of various normalization methods. Normalization was possible after loading and before training. Processed data was approximately 26.4 GB, and generated label data was approximately 3.60 MB.

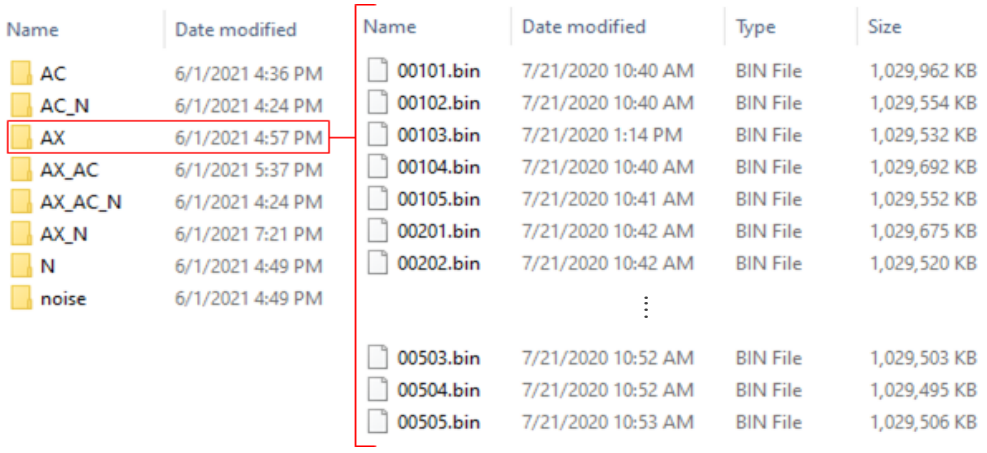

Figure 3-6. Directory structure and file naming for storage of raw data.

3.3.1 Sampling

After discarding the Q-component, the data contained approximately 527 million data points. The first 100 million data points and all data after 500 million was discarded. This resulted in a data array containing exactly 400 million I-components. A specified number of random samples were taken from the data with a large amount of space placed between each sample. The space could be characterized as both large and randomized to decrease correlation between samples. The number of samples taken from each file differed depending on data class.

Sample number was determined beforehand to ensure that sampled classes would be balanced (See Table 3-5). The intention was to later test multiple sample lengths in increments of 5,000 for determining ideal length. With this in mind, sample lengths taken during preprocessing were maximum size (i.e., 30,000) tested. Smaller sample lengths could be obtained by trimming samples. Using this method, only one dataset must be preprocessed and stored. Total samples were 118,125—16,875 per class.

Table 3-5. File Sampling Parameters by Class

Classes	Number Files	Number Samples	Max Space	Sample Length	Total Samples
0, 1, 2	25	675	570,000	30,000	16,875
3, 4, 5	125	135	2,900,000	30,000	16,875
6	135	125	3,100,000	30,000	16,875

3.3.2 Normalization

As previously mentioned, preprocessed data was stored in dBm so changing the normalization method would not require data reprocessing. Instead, preprocessed data could be loaded and normalized using any method before model training. Data was normalized in one of three ways: 1) offsetting by the threshold for duty cycle calculation in the LabVIEW software [16] and scaling to a range of [-1, 1] while preserving the new zero location, equation (3.1); 2) scaling to a range of [0, 1] using min-max normalization, equation (3.2); and 3) scaling to a range of [-1, 1] using min-max normalization, equation (3.3). All three normalization techniques were tested on early models, and normalization using equation (3.1) gave the best results. See Figure 3-7 below for examples of the three normalization methods. See Figure A-1in Appendix A, which shows random samples for all classes after (3.1) normalization.

$$x' = \frac{x + \Gamma}{\max(|\max(x)|, |\min(x)|)} \quad where, \Gamma\ is\ duty\ cycle\ threshold \tag{3.1}$$

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{3.2}$$

$$x' = 2\frac{x - \min(x)}{\max(x) - \min(x)} - 1 \tag{3.3}$$

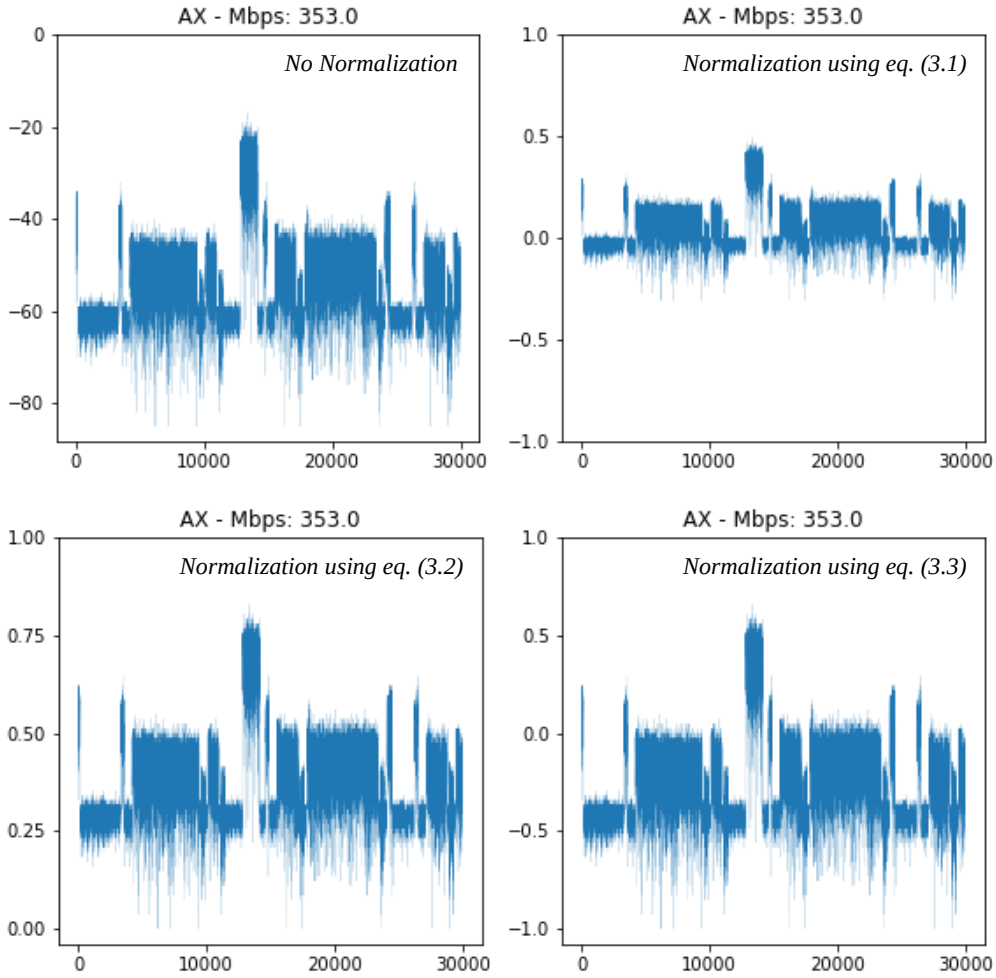

Figure 3-7. Samples of normalized data.

3.4 WIFINet – WaveNet Adapted Neural Network

A neural network was developed that adapted WaveNet's [1] original architecture, which took in text and synthesized raw audio speech from a distribution at each timestep. The adapted model, named WIFINet, maintained most of the architecture until after skip connections and ReLu activation. The first 1x1 convolution was replaced with a 32x1 convolution, and the second 1x1 convolution was replaced with additional layers to aid in classification. The Softmax also differed, in that, instead of outputting a distribution at each timestep, only one distribution was output, indicating to which of the seven classes the input most likely belonged. The input consisted of raw power measurement samples from only the in-phase component, since the quadrature component was discarded. Input data was preprocessed as described in Section 3.2, and no further data processing was done inside the neural network using preprocessing layers. Labels consisted of integer values in the range of one to seven for identifying the sample as 802.11ax, 802.11ac, 802.11n, 802.11ax with 802.11ac, 802.11ax with 802.11n, 802.11ac with 802.11n, and 802.11ax with 802.11ac with 802.11n, respectively. Labels were one-hot encoded. The TensorFlow 2.5 machine learning library for Python was used to build the model.

3.4.1 Architecture

The un-modified part of WIFINet consisted of a 1x1 convolution followed by a series of stacked residual blocks, which consists of a causal dilated convolution—the output of which was fed separately into both a sigmoid and tanh activation function. The sigmoid acted as a forget gate while the tanh acted as an activation. Sigmoid output and tanh were then multiplied together and input into another 1x1 convolution, which was added to the residual and transferred out of the block through a skip connection and was input to the next residual block. Since each residual

block had a skip connection, a series of skips emerged from the blocks. These skip connections were added together and they were input into an ReLu activation. At this stage, the WIFINet entered the modified portion, wherein the architecture began to differ from WaveNet [1].

In WIFINet's modified portion, the first 1x1 convolution was replaced with a 32x1 convolution. Next, the ReLu activation followed by a second 1x1 convolution was replaced with a series of layers consisting of a 32x1 average pooling layer; a flatten layer; a dense layer with 1,024 neurons and ReLu activation; a dropout layer at a rate of 0.5, a dense layer with 128 neurons and ReLu activation; another dropout layer at a rate of 0.5; and a dense layer with seven neurons and Softmax activation. The average pooling layer was used to decrease dimensionality. Pool size determines by what factor feature dimensions will be reduced. In this case presented herein, input was size (20000, 32), indicating 20,000 timesteps and 32 filters. Output was reduced to size (2000, 32) using a stride of 10. The flatten layer diminishes input to one dimension, so input size (2000, 32) would be flattened to 64000. The two dense layers use He uniform variance scaling as the kernel initializer. Two dropout layers were added for regularization which helps with overfitting during training. These layers will randomly set neurons to zero at a frequency of the specified rate at each step during training; 0.5 was used in WIFINet. See Figure 3-8, which depicts WIFINet. The model values are "optimal" values selected after tuning the model—the process described in greater detail in the next section.

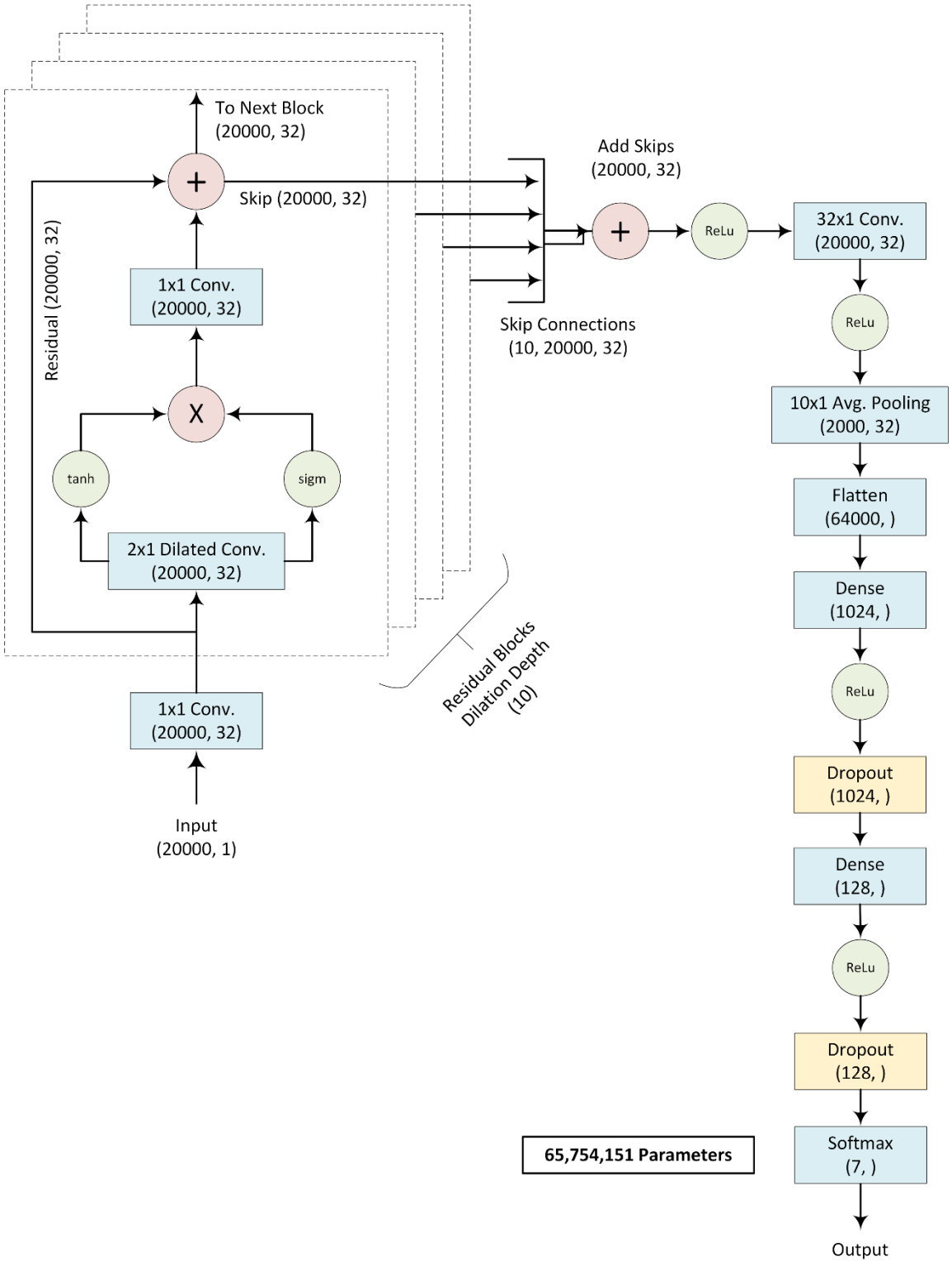

Figure 3-8. WIFINet architecture.

3.4.2 Optimization

Adam (i.e., Adaptive Moment Estimation) was used for the optimization algorithm. Unlike stochastic gradient descent (SGD), which uses the same learning rate throughout training—unless momentum or a learning rate decay scheduler is used—Adam computes adaptive learning rates for each parameter. Bias corrected first and second moments— $\hat{m}_t$ and $\hat{v}_t$,—of the gradients g_t are calculated using equations (3.4) and (3.5). [17]

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} = \frac{\beta_1 m_{t-1} + (1-\beta_1) g_t}{1-\beta_1^t} \tag{3.4}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} = \frac{\beta_2 m_{t-1} + (1-\beta_1) g_t^2}{1-\beta_2^t} \tag{3.5}$$

These moments are then used to update parameters using the Adam update formula shown in Equation (3.6). Moments $\hat{m}_t$ and $\hat{v}_t$ are used to calculate an adaptive learning rate η. Values β_1 and β_2 are decay rates with suggested default values of 0.9 and 0.999, respectively. Value ϵ is a small constant added for numerical stability with a suggested default value of $1e^{-8}$ from the original paper and $1e^{-7}$ from the Keras API. Suggested default values for decay rates β_1 and β_2 and the default value from Keras for ϵ were used. [17]

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t \tag{3.6}$$

3.4.3 Hyperparameter Tuning

WIFINet contains of number of network parameters for controlling training results, although many are not derived during training (e.g., neuron weights). Neuron weights typically proceed through a round of testing to determine optimal value (i.e., hyperparameter tuning). Although many approaches are available for hyperparameter tuning, the one utilized for

WIFINet involved testing a chosen set of values for each parameter, training the model with presets while maintaining others, and analyzing results after training was complete. When determining the optimal values, both loss and accuracy were taken into account, as well as training time. For instance, a change in accuracy by $+1.0\%$ would not necessarily be justified if training time was dramatically affected. After tuning each hyperparameter, two to three highest values from each were chosen, and one model for every combination was trained and evaluated. The best performing model was used to determine optimal hyperparameters for use in the final model training and evaluation. Table 3-1 shows each hyperparameter, tested values, and optimal value selected for the final model. Decay rates and stability constant parameters for the Adam optimizer used the default values and were not tuned.

Table 3-6. Hyperparameter Optimal and Tested Values

Parameter	**Value**	**Description**	**Tested Values**
Depth	10	Number of residual blocks.	2, 4, 6, 8, 10, 12
Filters	32	Number of convolution filters.	16, 32, 64, 128
K_1	1	Kernel for input convolution.	1, 2
K_2	32	Kernel for last convolution.	1, 3, 5, 7, 16, 32, 50, 64, 128
Stride	10	Average pooling stride size.	10, 25, 50, 100
D_1	1024	Size of 1st dense layer.	16, 32, 64, 128, 256, 512, 1024, 2048
D_2	128	Size of 2nd dense layer.	16, 32, 64, 128, 256, 512, 1024, 2048
α	1e-4	Adam learning rate.	1e-3, 1e-4, 1e-5, 1e-6
β_1	0.9	Adam decay rate - 1st moment.	-
β_2	0.999	Adam decay rate - 2nd moment.	-
ϵ	1e-7	Adam stability constant.	-
Epochs	10	Passes over data for training.	-
Batch	32	Samples processed before update.	-

In addition to hyperparameter tuning, sampled data size (i.e., number of timesteps) was also tested. As explained in Section 3.3.1, data was preprocessed with the sample size of 30,000, which was the maximum amount for testing. Smaller sizes could be created by simply trimming the end of the data to the desired size. This involved preprocessing and saving only one dataset rather than one for each desired testing size. Sizes tested ranged from 5,000 to 30,000 in increments of 5,000. Figure 3-9 shows testing results; 20,000 was selected, as little improvement was demonstrated beyond this size.

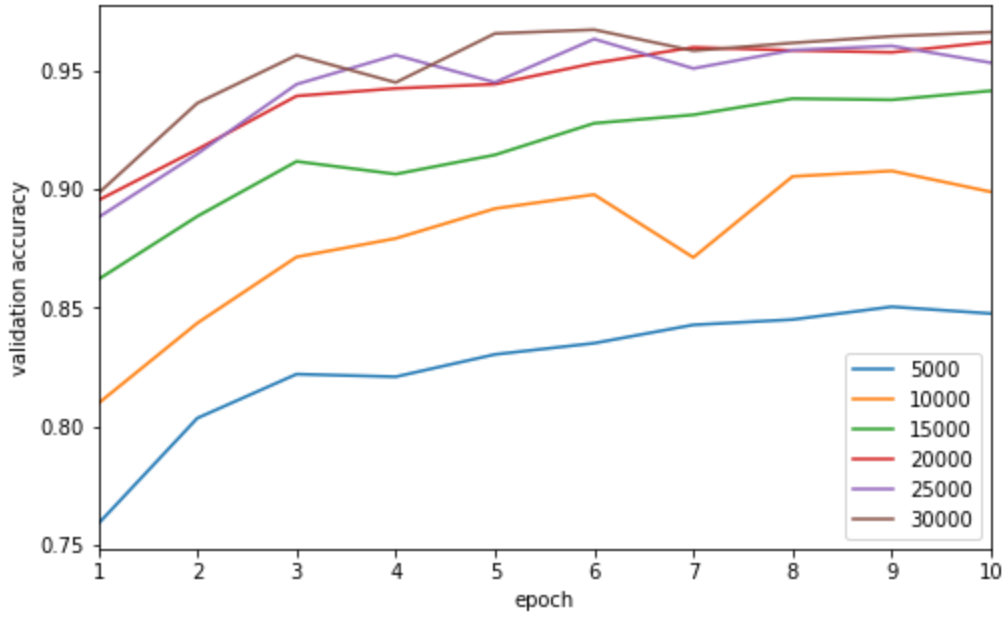

Figure 3-9. WIFINet results for various sample sizes.

3.4.4 Hardware and Source Code

Data preprocessing and model training was performed on a desktop computer running Windows 10 Pro with an AMD Ryzen 9 5900X 12-core 3.70 GHz processor, 32 GB of DDR4 Dram, and an NVIDIA GeForce RTX 3070 GPU. Python version 3.8.8 was utilized for all code. Source code can be found on GitHub [18].

3.4.5 WIFINet Results

Data was split into a training and validation set ratio of 5:1 (i.e., training set was 80% and validation set was 20%). Training ran for 10 epochs, and the model was set to monitor validation loss and save the optimal weights. For example, if the next epoch resulted in a validation loss increase, weights were not saved. This process took approximately 2.88 hours with a training sample size of 94,500. A validation sample size of 23,625 required approximately 77.29 seconds (i.e., 1.29 minutes) for the model to process. Prediction time for one sample was 0.003 seconds. Table 3-7 shows accuracy, precision, recall, and F-score calculations, and Figure 3-10 shows training loss vs. validation loss, as well as training accuracy vs. validation accuracy. The graphs demonstrate that validation loss reaches a minimum of 0.0699 at epoch 8, which corresponds to 98.10% validation accuracy. Weights at this epoch were retained for the final model since validation loss was monitored as described above.

Table 3-7. WIFINet Metrics

Accuracy	**Precision**	**Recall**	**F-Score**
0.980995	0.981016	0.980995	0.980993

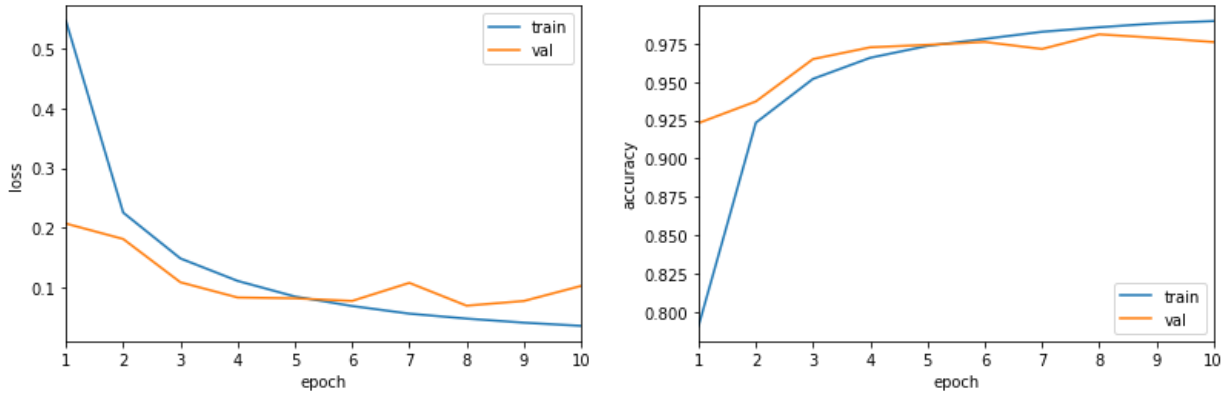

Figure 3-10. WIFINet training loss and accuracy.

Figure 3-11 depicts the confusion matrix for the validation set. Each row represents the actual sample class, and each column represents model-predicted class for the sample. Model accuracy is separated for each class, most of which were correctly predicted 98% of the time. 802.11ax with 802.11ac and 802.11ax with 802.1 were the exceptions, as they were correctly predicted 97% and 96%, respectively. Figure 3-12 shows the receiver operating characteristic (ROC) curve and the area under the curve (AUC) calculation for each class, as well as micro and macro averaging.

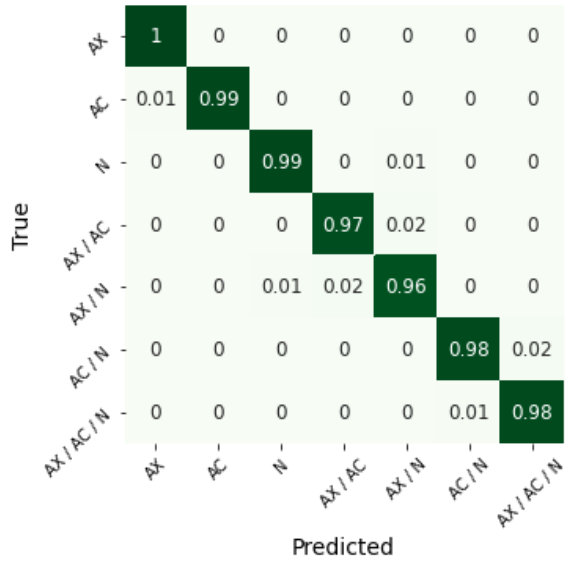

Figure 3-11. WIFINet confusion matrix.

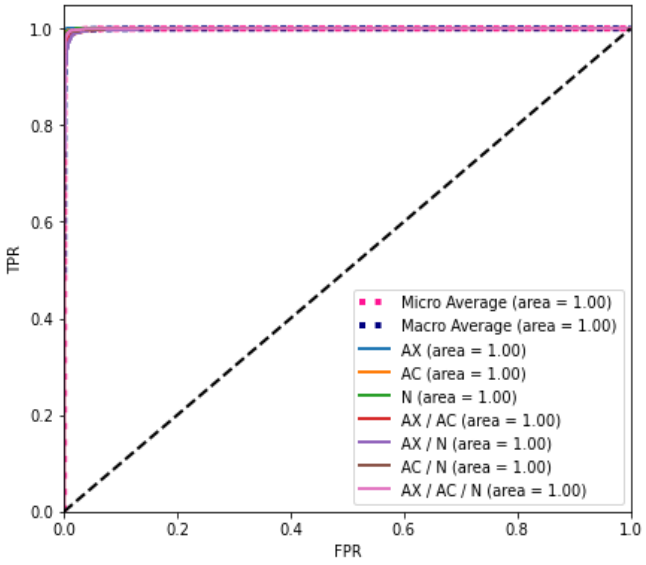

Figure 3-12. WIFINet ROC curve.

Chapter 4 Additional Machine Learning Models to Classify Wi-Fi

Neural network and traditional machine learning models were trained on the same dataset for the sake of comparison. Data preprocessing was unchanged. More importantly normalization was unchanged. It should be noted that in some instances various normalization methods could have yielded better results for some models. Additionally, while all models had some form of tuning, time did not permit in depth tuning to the extent it was for WIFINet. As such, more in depth tuning for these models could yield better results as well.

4.1 Basic Artificial Neural Network (ANN)

A basic artificial neural network (See Figure 4-2)—one containing only layers of perceptrons or artificial neurons—was trained and tested with multiple sizes of two-to-three hidden layers with dropout rate of 0.5 used between layers. Optimization and parameters were the same as those used for WIFINet. Test results are shown in Figure 4-1. Only the best-performing model that used three hidden layers of sizes [2048, 512, 128] was retained. Metrics calculations are shown in Table 4-1. The model was trained in approximately 225 seconds (i.e., 3.75 minutes), and prediction time was approximately 1.25 seconds for all samples and 0.002 seconds for one sample. The confusion matrix and ROC curve are shown in Figure 4-3 and Figure 4-4. The TensorFlow 2.5 machine learning Python library was used to build the model.

Table 4-1. ANN Metrics

Accuracy	Precision	Recall	F-Score
0.721101	0.721944	0.721101	0.713134

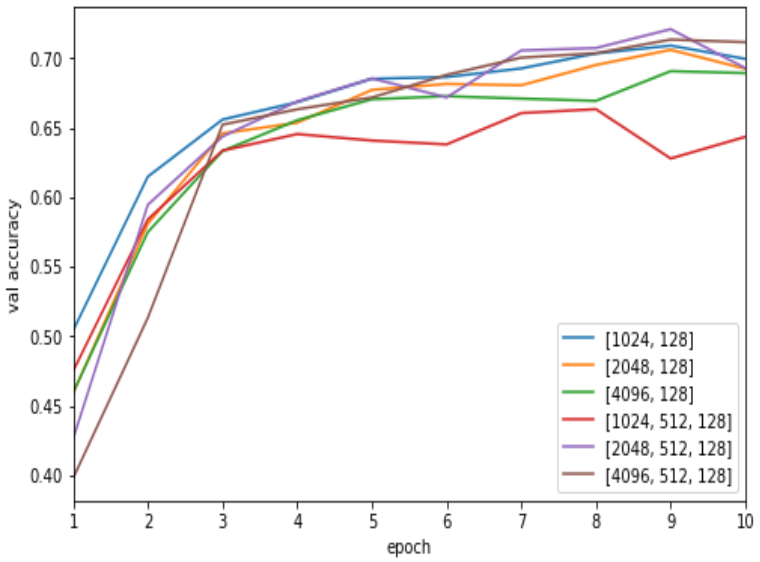

Figure 4-1. ANN results for various depths and sizes.

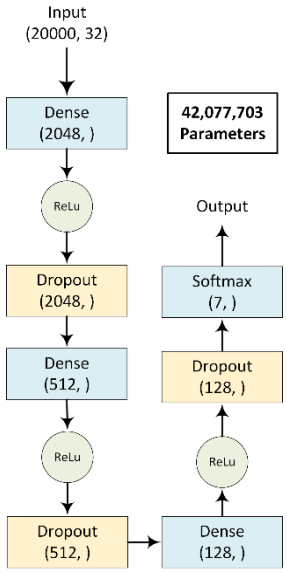

Figure 4-2. ANN structure.

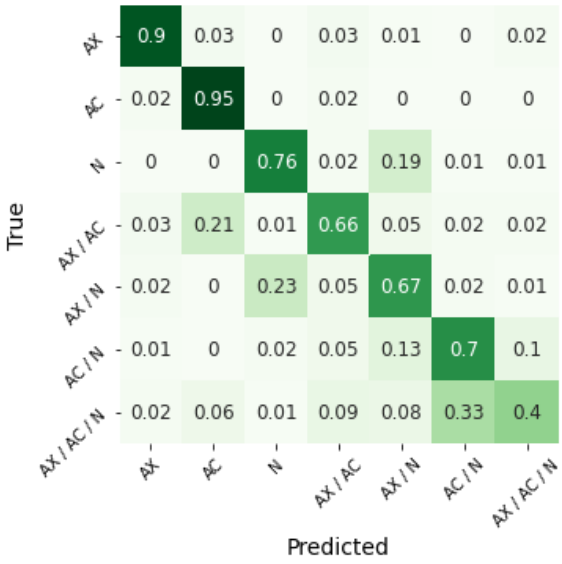

Figure 4-3. ANN confusion matrix.

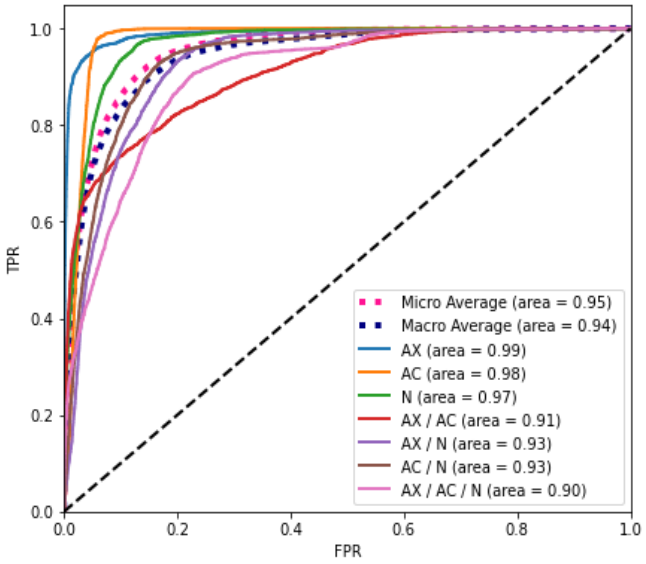

Figure 4-4. ANN ROC curve.

4.2 Convolutional Neural Network (CNN)

A convolutional neural network (See Figure 4-6) was trained and tested for various depths. The structure consisted of three groups of convolution layers with an average pooling layer after each group down-sampling by a factor of 2. Filter sizes 16, 32, and 64 and kernel sizes 3, 5, and 3 were used. Output layers remained the same as for WIFINet and consisted of a flatten layer followed by two dense layers with dropout rate of 0.5 after each layer, and then a Softmax layer at the output. The same optimization and parameters used for WIFINet were employed. Results for each test are shown in Figure 4-5. Depths 1, 3, and 2 outperformed others and were retained. Metrics calculations are shown in Table 4-2. The model was trained in approximately 1,395 seconds (i.e., 23.25 minutes), and its prediction time was approximately 8.53 seconds for all samples and 0.003 seconds for one sample. The confusion matrix and ROC curve are shown in Figure 4-7 and Figure 4-8. The TensorFlow 2.5 machine learning Python library was used to build the model.

Table 4-2. CNN Metrics

Accuracy	**Precision**	**Recall**	**F-Score**
0.963302	0.963356	0.963302	0.963298

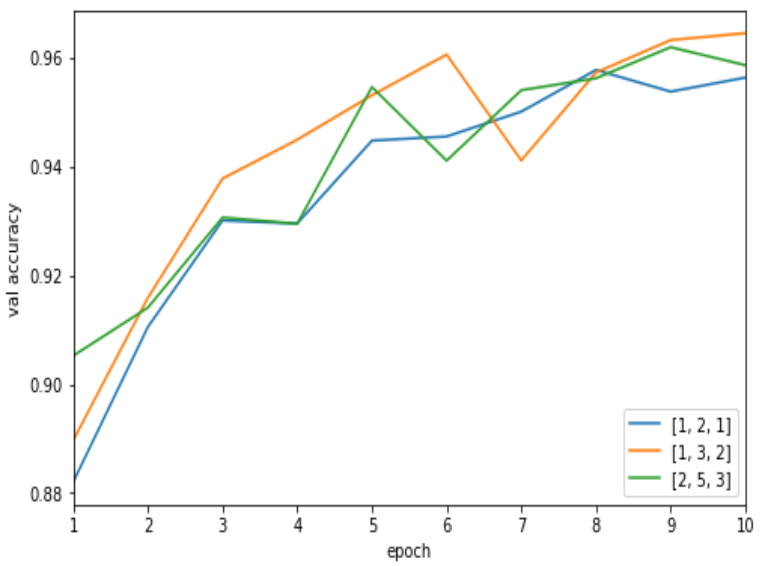

Figure 4-5. CNN results for various depths.

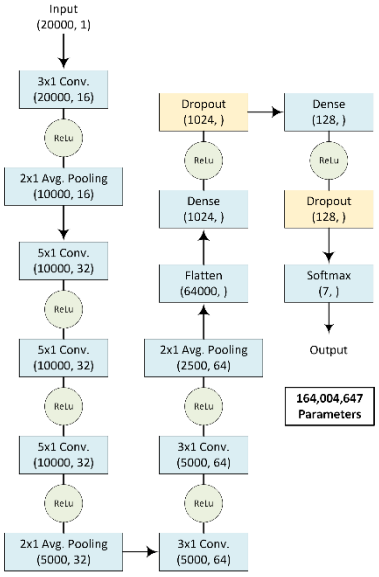

Figure 4-6. CNN structure.

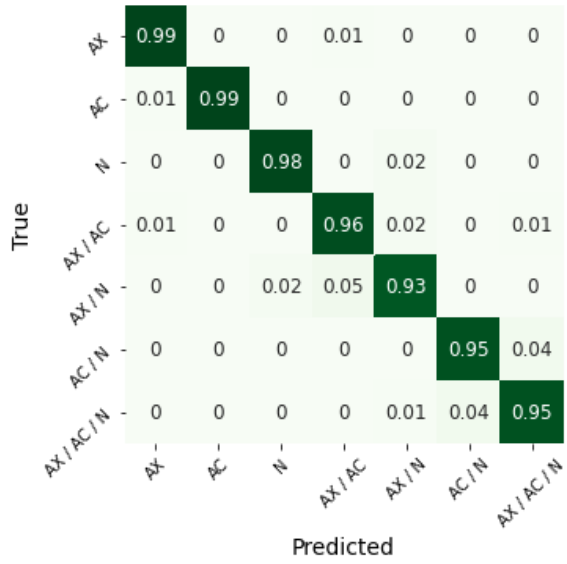

Figure 4-7. CNN confusion matrix.

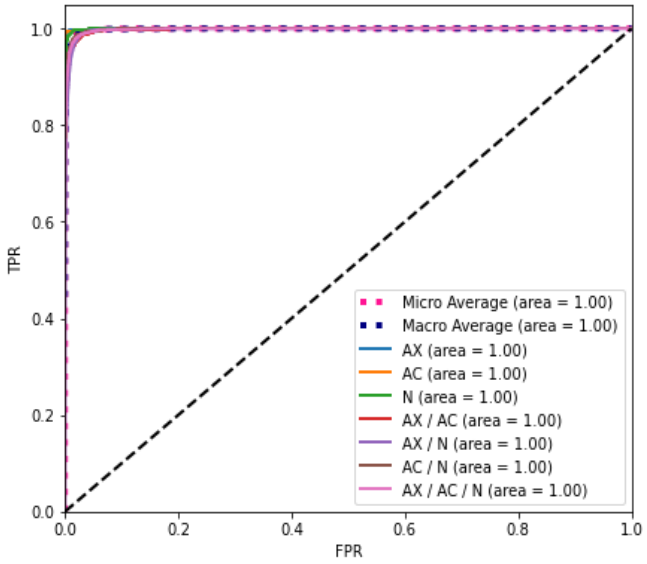

Figure 4-8. CNN ROC curve.

4.3 K-Nearest Neighbors (KNN) Classifier

A k-nearest neighbor classifier was trained and tested for various numbers of neighbors. Unlike training for other algorithms, this algorithm utilizes multiple cores when predicting and was set to use the maximum number. All other parameters were left as default. Results for each test are shown in Figure 4-9. The classifier with three neighbors outperformed others and was retained. Metrics calculations are shown in Table 4-3. The model took approximately 507.09 seconds (i.e., 8.45 minutes) to train, and its prediction time was approximately 9254.25 seconds (i.e., 2.57 hours) for all samples and 0.194 seconds for one sample. The confusion matrix and ROC curve are shown in Figure 4-10 and Figure 4-11. The scikit learn 0.23.2 Python library was used to build the classifier.

Table 4-3. KNN Classifier Metrics

Accuracy	Precision	Recall	F-Score
0.523175	0.534303	0.523175	0.513713

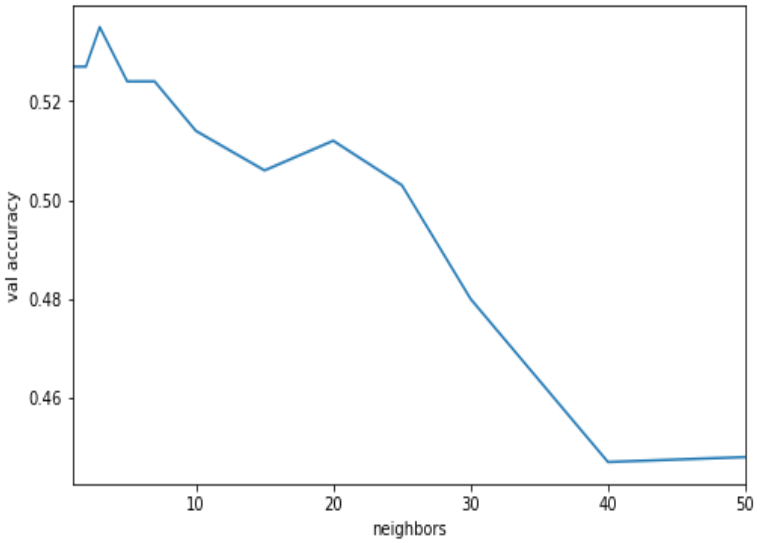

Figure 4-9. KNN results for various number of neighbors.

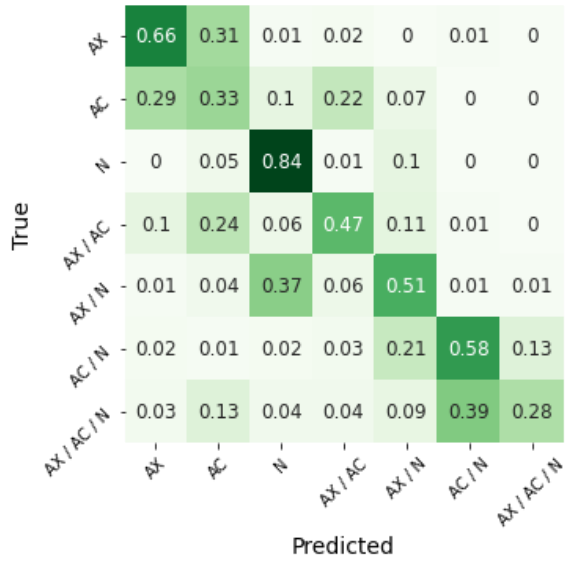

Figure 4-10. KNN confusion matrix.

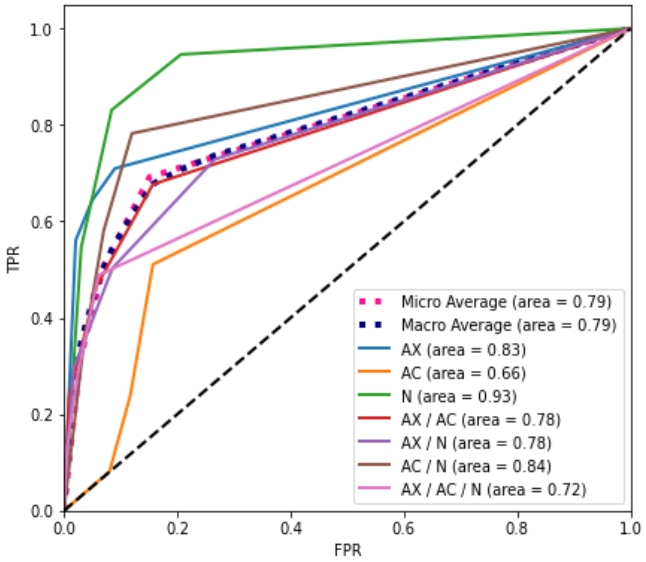

Figure 4-11. KNN ROC curve.

4.4 Support Vector Machine (SVM) Classifier

A support vector machine classifier was trained. A linear kernel was utilized, as well as probability calibration using 5-fold cross validation. All other parameters were left as default. Metrics calculations are shown in Table 4-4. The model took approximately 63723.88 seconds (i.e., 17.70 hours) to train, and its prediction time was approximately 6.61 seconds for all samples and less than 0.001 seconds for one sample. The confusion matrix and ROC curve are shown in Figure 4-12 and Figure 4-13. The scikit learn 0.23.2 Python library was used to build the classifier.

Table 4-4. SVM Classifier Metrics

Accuracy	Precision	Recall	F-Score
0.226963	0.219781	0.226963	0.186025

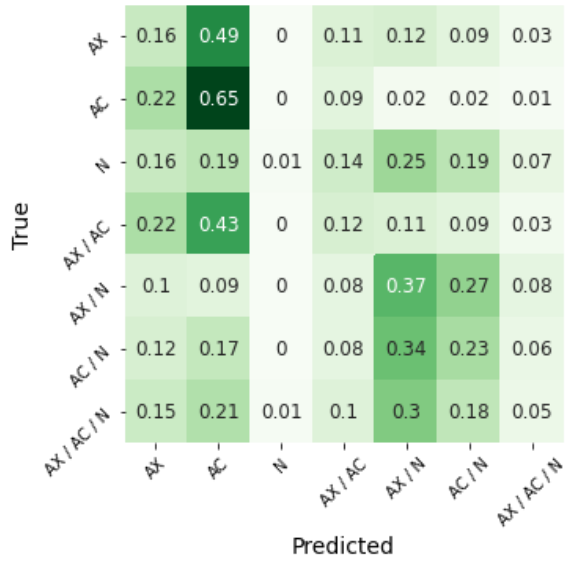

Figure 4-12. SVM confusion matrix.

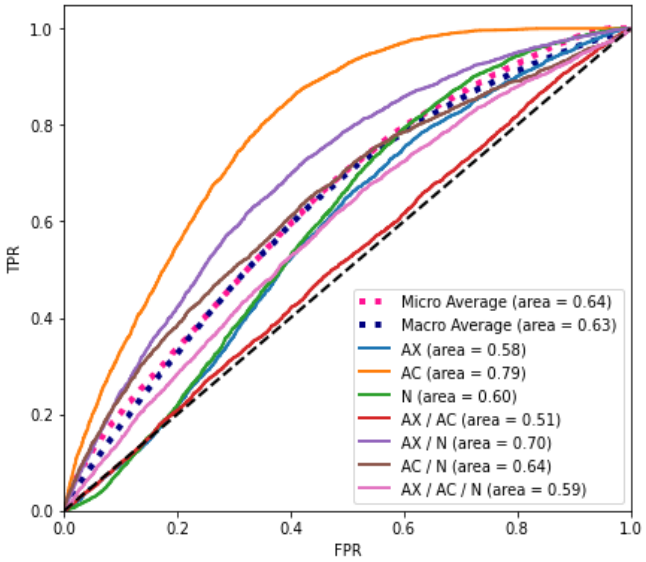

Figure 4-13. SVM ROC curve.

4.5 Decision Tree Classifier

A decision tree classifier was trained and tested for various numbers of minimum leaves required for a leaf node. All other parameters were left as default. Results of each test are shown in Figure 4-14. The 25-leave test outperformed others and was retained. Metrics calculations are shown in Table 4-5. The model took approximately 1056.72 seconds (i.e., 17.61 minutes) to train, and its prediction time was approximately 0.47 seconds for all samples and less than 0.001 seconds for one sample. The confusion matrix and ROC curve are shown in Figure 4-15 and Figure 4-16. The scikit learn 0.23.2 Python library was used to build the classifier.

Table 4-5. Decision Tree Metrics

Accuracy	Precision	Recall	F-Score
0.531259	0.526679	0.531259	0.528002

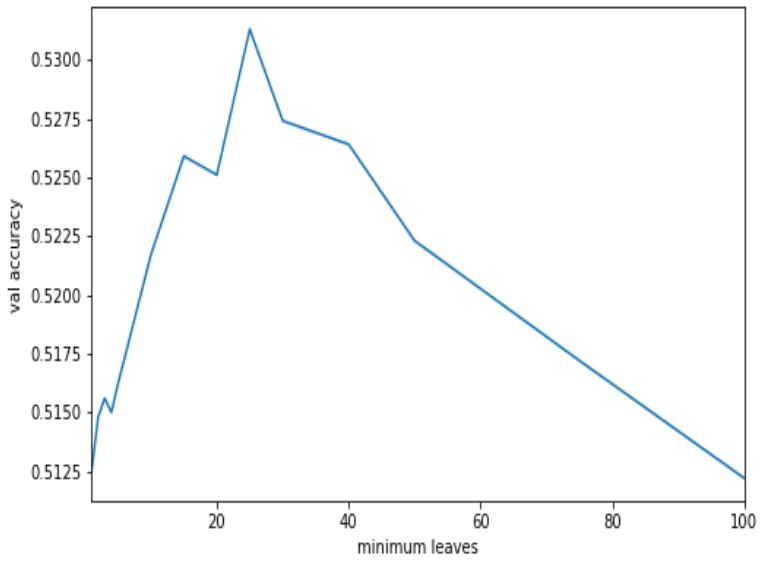

Figure 4-14. Decision tree results for various number of minimum leaves.

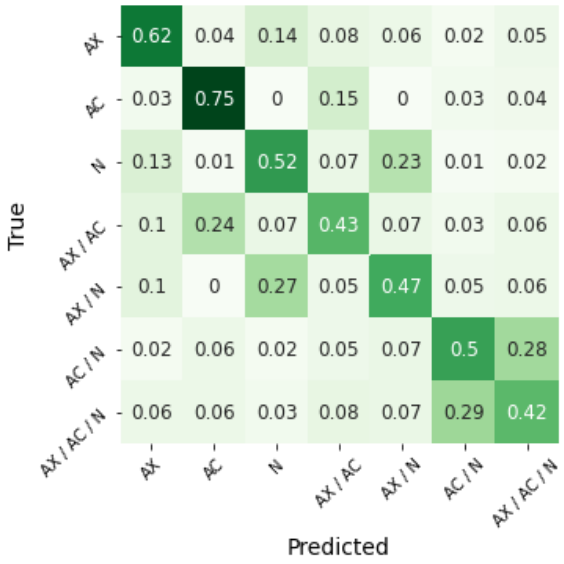

Figure 4-15. Decision tree confusion matrix.

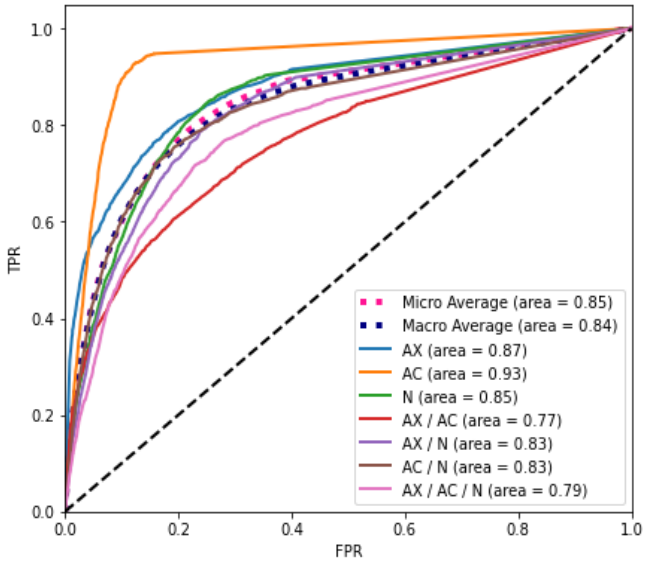

Figure 4-16. Decision tree ROC curve.

4.6 Random Forest Classifier

A random forest classifier—an ensemble of multiple decision trees—was trained and tested for various numbers of trees. The algorithm utilizes multiple cores, and this was set to use the maximum number. All other parameters were left as default. The results of each test are

shown in Figure 4-17. The test with 100 trees outperformed others and was retained. Metrics calculations are shown in Table 4-6. The model took approximately 8632.03 seconds (i.e., 2.40 hours) to train, and its prediction time was approximately 0.60 seconds for all samples and 0.001 seconds for one sample. The confusion matrix and ROC curve are shown in Figure 4-18 and Figure 4-19. The scikit learn 0.23.2 Python library was used to build the classifier.

Table 4-6. Random Forest Classifier Metrics

Accuracy	**Precision**	**Recall**	**F-Score**
0.746624	0.75821	0.746624	0.739329

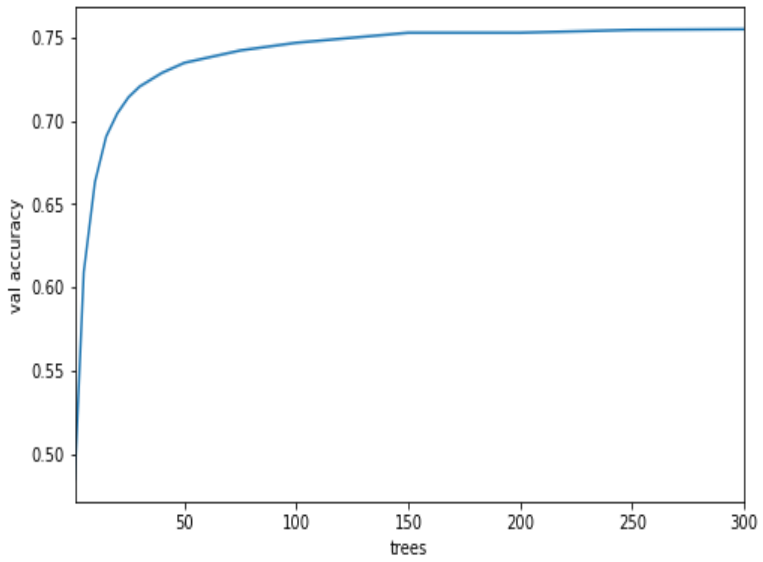

Figure 4-17. Random forest results for various number of trees.

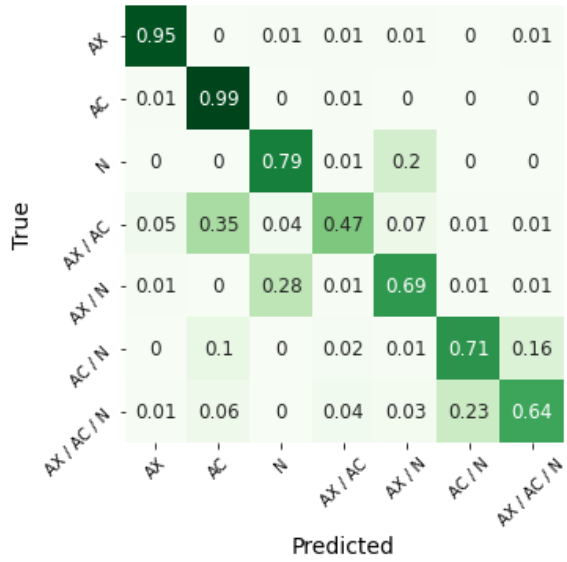

Figure 4-18. Random forest confusion matrix.

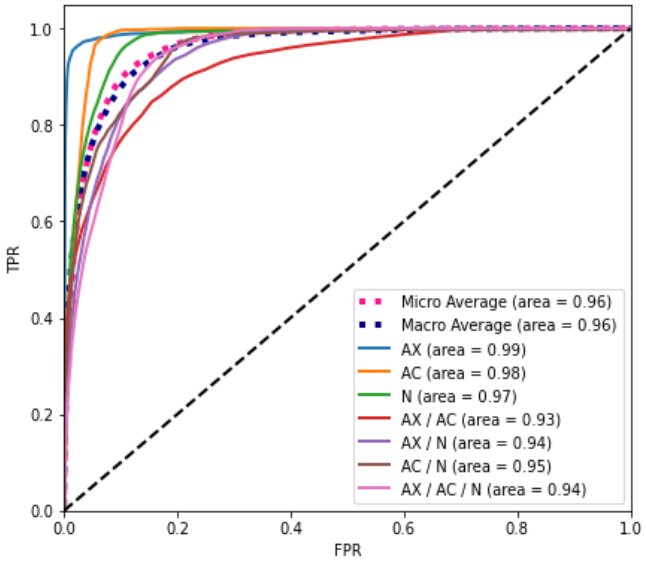

Figure 4-19. Random forest ROC curve.

4.7 Gaussian Naïve Bayes Classifier

A gaussian naïve Bayes classifier was trained and tested. All parameters were left as default. Metrics calculations are shown in Table 4-7. The model took approximately 12.10 seconds to train, and its prediction time was approximately 23.85 seconds for all samples and less than 0.001 seconds for one sample. The confusion matrix and ROC curve are shown in Figure 4-20 and Figure 4-21. The scikit learn 0.23.2 Python library was used to build the classifier.

Table 4-7. Naïve Bayes Classifier Metrics

Accuracy	Precision	Recall	F-Score
0.447958	0.433623	0.447958	0.414635

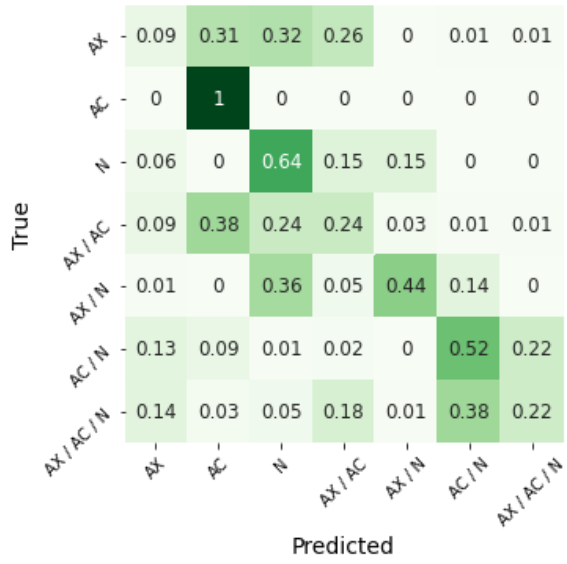

Figure 4-20. Naïve bayes confusion matrix.

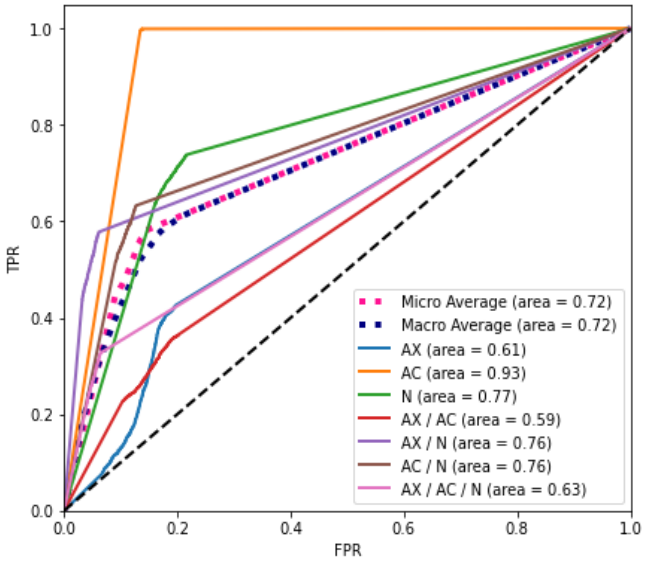

Figure 4-21. Naïve bayes ROC curve.

4.8 Logistic Regression Classifier

A logistic regression classifier was trained and tested. The algorithm utilizes multiple cores and was set to use the maximum number. All parameters were left as default. Its metrics calculations are shown in Table 4-8. The model took approximately 3014.04 seconds (i.e., 50.23 minutes) to train, and its prediction time was approximately 0.4 seconds for all samples and less than 0.001 seconds for one sample. The confusion matrix and ROC curve are shown in Figure 4-22 and Figure 4-23. The scikit learn 0.23.2 Python library was used to build the classifier.

Table 4-8. Logistic Regression Classifier Metrics

Accuracy	Precision	Recall	F-Score
0.232423	0.233407	0.232423	0.232781

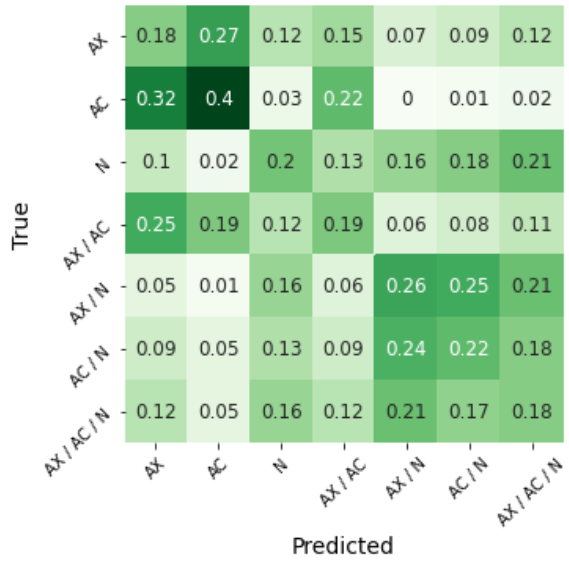

Figure 4-22. Regression confusion matrix.

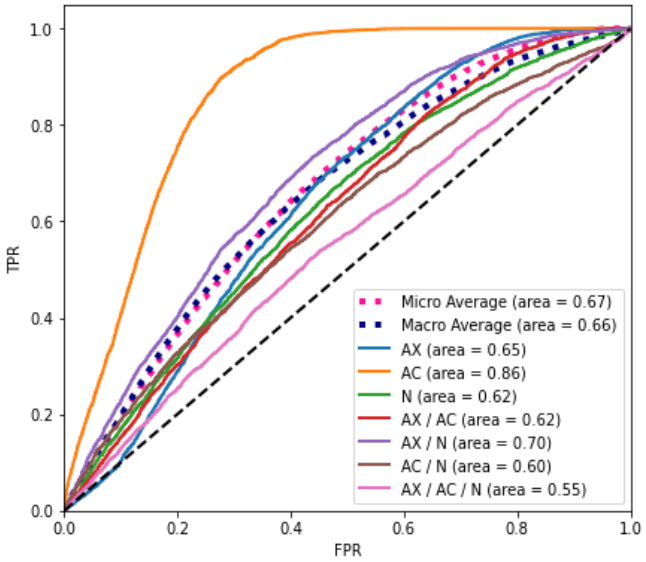

Figure 4-23. Regression ROC curve.

4.9 AdaBoost Classifier

An AdaBoost classifier was trained and tested. All parameters were left as default. Metrics calculations are shown in Table 4-9. The model took approximately 7722.32 seconds (i.e., 2.15 hours) to train, and its prediction time was approximately 30.60 seconds for all samples and 0.001 seconds for one sample. The confusion matrix and ROC curve are shown in Figure 4-25 and Figure 4-26. The scikit learn 0.23.2 Python library was used to build the classifier.

Table 4-9. AdaBoost Classier Metrics

Accuracy	Precision	Recall	F-Score
0.393735	0.366133	0.393735	0.327122

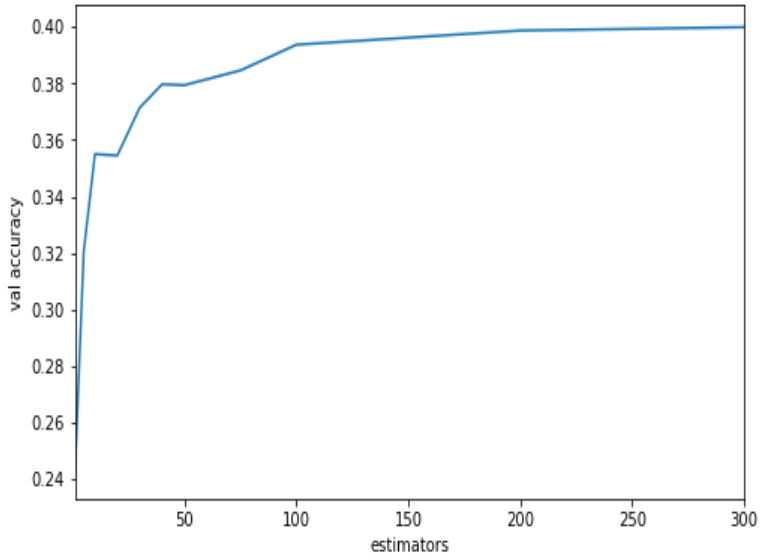

Figure 4-24. AdaBoost results for various number of estimators.

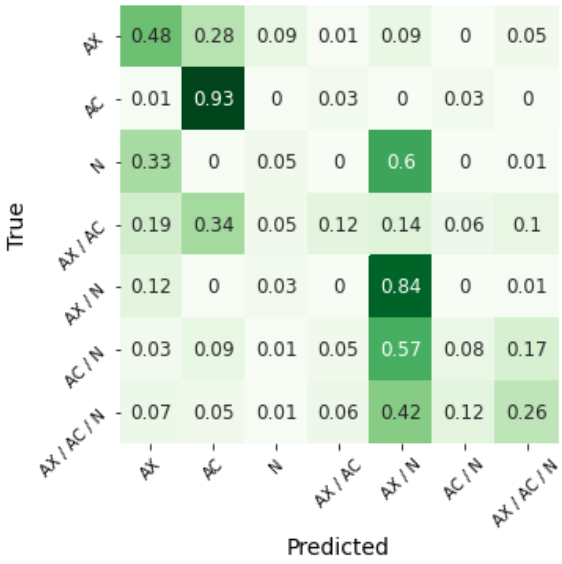

Figure 4-25. AdaBoost confusion matrix.

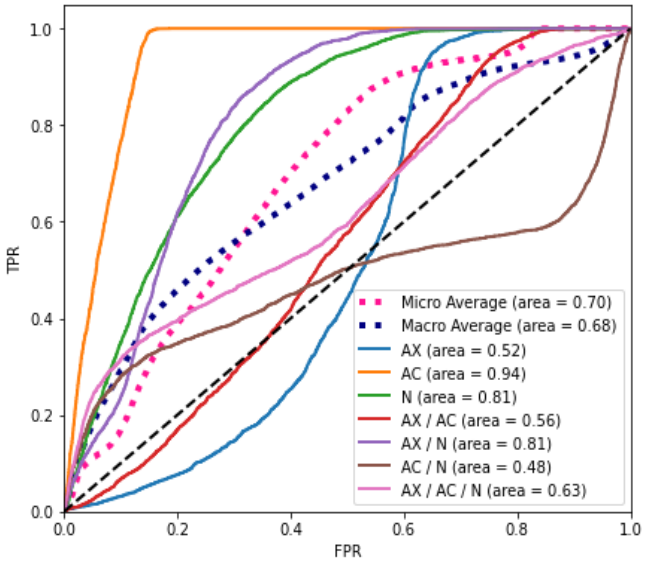

Figure 4-26. AdaBoost ROC curve.

Chapter 5 Evaluation and Comparison of Models

5.1 Metric Evaluation

Metrics for all models are listed in Table 5-1. WIFINet outperformed others with metrics reported at 98.1%. The CNN model had similar performance to WIFINet with metrics at 96.3%. ANN and Random Forest models had similar performances with metrics ranging between 72 – 76%—far behind WIFINet and CNN model performance. Figure 5-1 plots all model accuracies for easy comparison. Figure 5-2 shows the ROC curves for all of the models. The top is the micro-averaged ROC curves and the bottom is the macro-averaged ROC curves. The right plots are zoomed in to the top-left corner in order to better compare differences between the top models. Like the metrics, ROC curves show WIFINet outperformed others and was followed closely behind by the CNN model. The remainder fell far behind.

Table 5-1. Model Metrics

Model	Accuracy	Precision	Recall	F-Score
WIFINet	**0.980995**	**0.981016**	**0.980995**	**0.980993**
ANN	0.721101	0.721944	0.721101	0.713134
CNN	0.963302	0.963356	0.963302	0.963298
KNN	0.523175	0.534303	0.523175	0.513713
SVM	0.226963	0.219781	0.226963	0.186025
Tree	0.531259	0.526679	0.531259	0.528002
Forest	0.746624	0.758210	0.746624	0.739329
Naïve Bayes	0.447958	0.433623	0.447958	0.414635
Log. Regression	0.232423	0.233407	0.232423	0.232781
AdaBoost	0.393735	0.366133	0.393735	0.327122

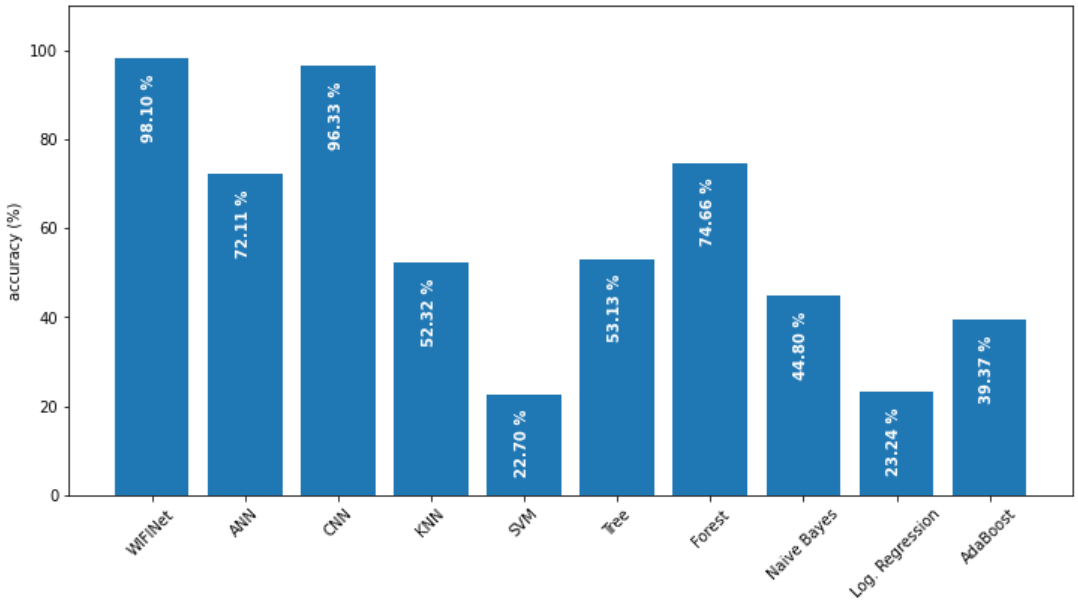

Figure 5-1. Comparison of accuracies.

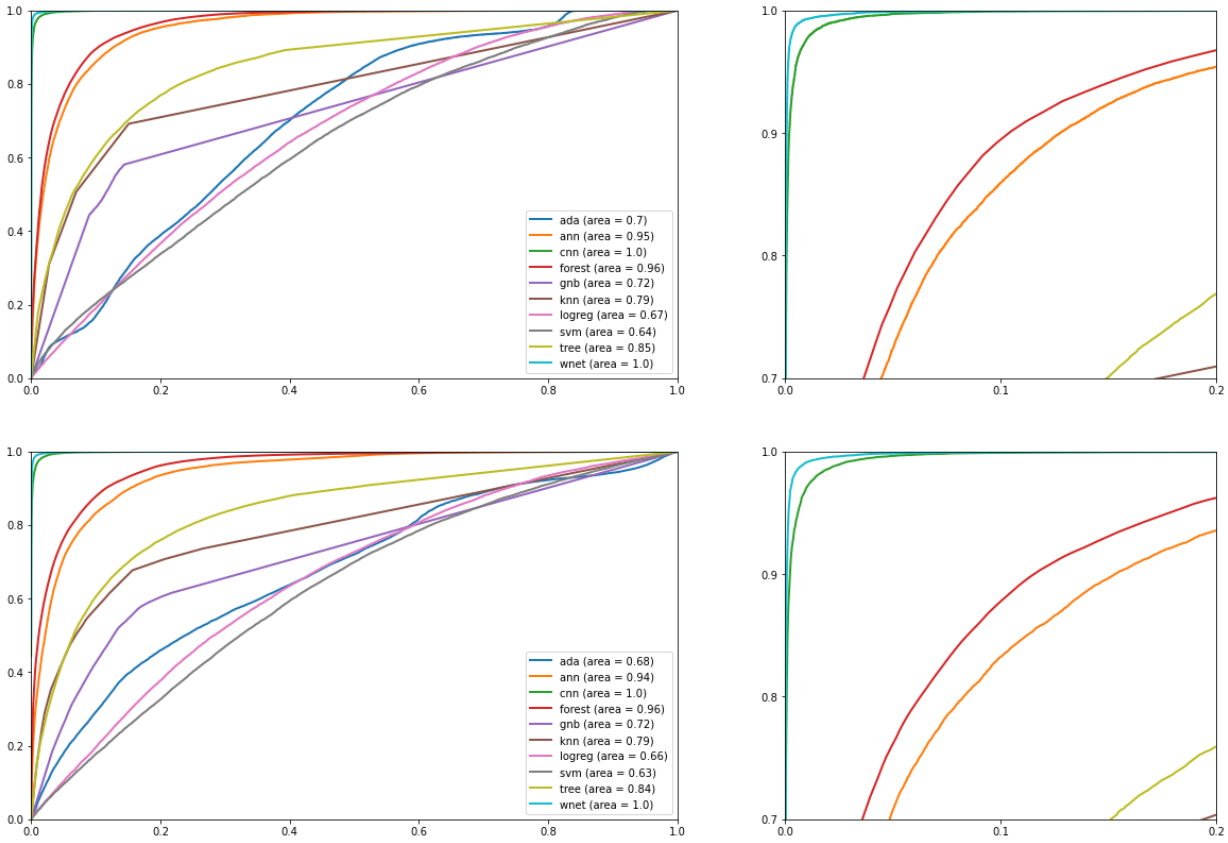

Figure 5-2. Comparison of micro (top) and macro (bottom) ROC curves.

5.2 Time and Size Evaluation

When comparing model time performance and size—two important attributes depending on the intended application—models vary drastically. Training and predicting times and model sizes are listed in Table 5-2.The Naïve bayes model took around 12.10 seconds to train, while other models took minutes—even hours. The SVM model took the longest to train—63,728.88 seconds (i.e., 17.10 hours). The logistic regression model took only 0.40 seconds to predict all 23,625 validation samples. KNN processing took the longest—9,254.25 seconds (i.e., 2.57 hours). Most importantly, the time to predict one sample for all models was comparable. Most took $\leq$ 0.003 seconds, except KNN, which took 0.194 seconds. Figure 5-3 and Figure 5-4 plot model times and sizes, respectively, for easy comparison.

Table 5-2. Model Times and Sizes

Model	Time (s)			Size (kB)
	Training	Prediction	Prediction	
WIFINet	10,371.00	77.29	0.003	770,889
ANN	225.00	1.25	0.002	493,144
CNN	1,395.00	8.53	0.003	1,922,018
KNN	507.09	9,254.25	0.194	23,429,362
SVM	63,723.88	6.61	**< 0.001**	5,474
Tree	1,056.72	0.47	**< 0.001**	566
Forest	8,632.03	0.60	0.001	186,005
Naïve Bayes	**12.10**	23.85	**< 0.001**	2,189
Log. Regression	3,014.04	**0.40**	**< 0.001**	548
AdaBoost	7,722.32	30.60	0.001	**66**

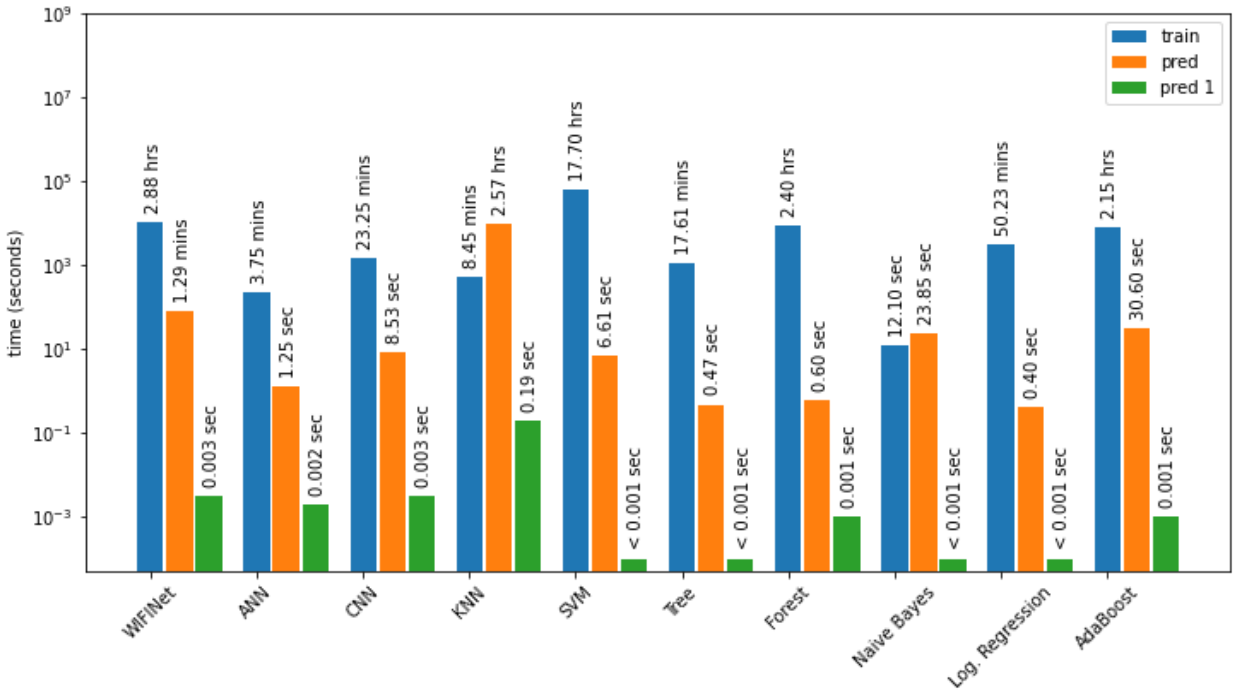

Figure 5-3. Comparison of times.

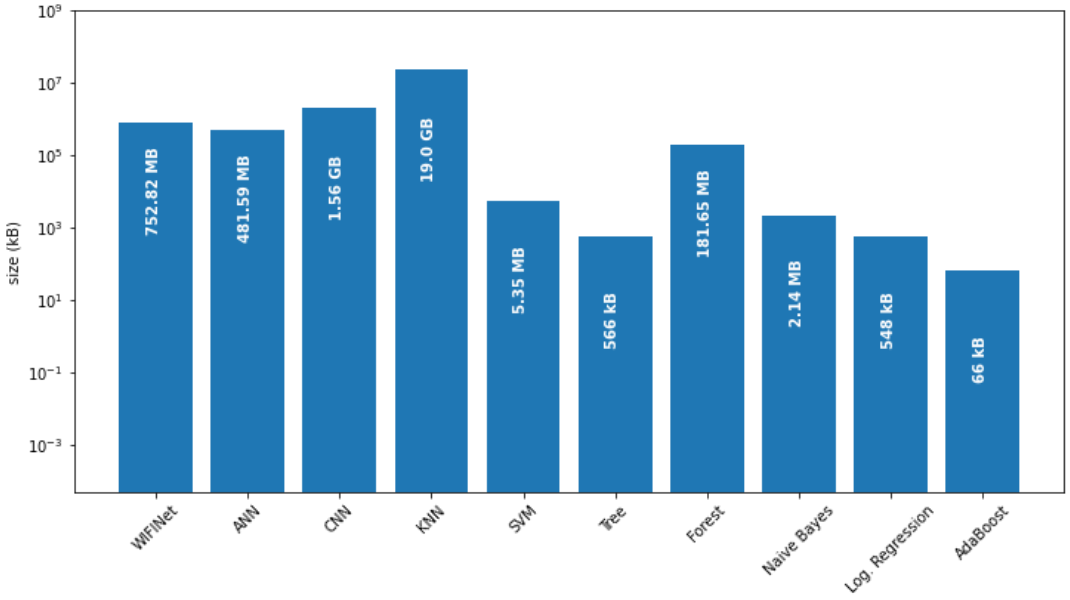

Figure 5-4. Comparison of sizes

5.3 Summary of Model Evaluation

WIFINet and CNN models exhibited acceptable performance for classifying coexisting wireless technologies. The WIFINet model achieved the superior 98.1% accuracy, and CNN achieved 96.3% accuracy. The WIFINet model took longer to train—10,371.00 seconds (i.e., 2.88 hours); the CNN model took only 1,395.00 seconds (i.e., 23.25 minutes). When predicting 23,625 samples in bulk, the WIFINet model took 77.29 seconds (i.e., 1.29 minutes), while the CNN took 8.53 seconds. Both WIFINet and CNN had comparable times when predicting only one sample—0.003 and 0.002 seconds, respectively. At two to three milliseconds, both of these models are sufficiently fast enough at predicting one sample that they can be implemented in real-time applications.

Another important characteristic one must consider is overall model size. For instance, if the model is be deployed on an embedded system, or any system where memory is limited, then model size becomes a deciding factor when choosing a model. WIFINet is 770,889 kB (i.e., 752.82 MB), while CNN is more than double the size—1,922,018 kB (i.e., 1.56 GB).

In most use cases, the time to predict one sample would be one of the most important characteristics to consider, along with overall accuracy. If this was the case, or if deploying on a memory-limited system, then the WIFINet model would be preferable. However, if fast training time or efficient bulk-prediction capabilities are desired, then the CNN model would probably be the better choice, given that a small loss in accuracy is permissible.

Chapter 6 Conclusion

6.1 Summary and Conclusions

The work presented in this thesis hypothesized that a neural network using dilated convolutions similar to WaveNet [1] would excel at coexisting wireless technology classification, namely WIFINet. Raw power measurements were collected at an IQ sampling rate of 10 MS/s in an environment with coexisting wireless technologies that spanned the 802.11n, 802.11ac and 802.11ax wireless specifications. The I-component was used to train the WIFINet model, using 80% of the data for sampling and 20% for validation. The model implemented stacked residual blocks with dilated convolutions and skip connections. WIFINet was demonstrated to accurately classify coexisting wireless technologies at an accuracy, precision, recall, and F-score above 98% (See Table 3-7, Table 5-1). Additional models, both neural networks and traditional machine learning methods, were created to compare WIFINet to models without dilated convolutions. Additional models performed poorly, with the exception of the CNN neural network, which utilized convolutions that were not dilated. This model's performance was similar to WIFINet with metrics at or approximate to 96% (See Table 4-2, Table 5-1). Results indicate that although WIFINet achieved the highest accuracy, exceptional performance was not likely directly related to dilated convolutions used in the model. In other words, results indicates that the generalization of sampled data could be made by both WIFINet and CNN models by generalizing short-range temporal patterns. This outcome is discerning because the main purpose for dilated convolution in WIFINet is its ability to model long-range temporal patterns; notably, CNN did not contain dilated convolutions and still performed similarly to WIFINet.

6.2 Future Work

This thesis reported a process for classifying coexisting wireless technologies using a neural network that utilized dilated convolutions to model long-range temporal patterns. Additional 802.11 specifications, as well as other wireless technologies like Bluetooth, Zigbee, RFID, or cellular (e.g., 3G, 4G, 5G, LTE) should be included in future work. It is entirely possible that although the dilated convolutions had little impact in this study, long-range temporal patterns might become more important when examining coexisting wireless technologies in more dense and more varied environments. Additional future work could collect additional data in less "ideal" environments to test the model's ability to classify samples with various types of noise. Researchers should keep in mind the importance of testing whether a model is effectively generalized to the degree at which it could classify noisy real-world samples without requiring additional training for each type of environment.

References

[1] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior and K. Kavukcuoglu, "WaveNet: A Generative Model for Raw Audio," in *The 9th ISCA Speech Synthesis Workshop*, Sunnyvale, 2016.

[2] Cisco, "Cisco Annual Internet Report (2018–2023) White Paper," 9 March 2020. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html. [Accessed 1 7 2021].

[3] M. Andrews, "DeepMind's WaveNet : How it works, and how it is evolving - TensorFlow and Deep Learning," Engineers.SG (YouTube Channel), 23 January 2018. [Online]. Available: https://www.youtube.com/watch?v=YyUXG-BfDbE. [Accessed 15 June 2020].

[4] J. Miller, "When Recurrent Models Don't Need to be Recurrent," The Berkeley Artificial Intelligence Research (BAIR) Lab, 6 August 2018. [Online]. Available: https://bair.berkeley.edu/blog/2018/08/06/recurrent/. [Accessed 15 June 2020].

[5] J. Singh, "WaveNet: Google Assistant's Voice Synthesizer.," Towards Data Science Inc., 7 November 2018. [Online]. Available: https://towardsdatascience.com/wavenet-google-assistants-voice-synthesizer-a168e9af13b1. [Accessed 15 June 2020].

[6] J. Boilard, P. Gournay and R. Lefebvre, "A Literature Review of WaveNet: Theory, Application and Optimization," in *146th Convention of the Audio Engineering Society*, Dublin, 2019.

[7] M. Thiel, "An Intuitive Introduction to Deep Autoregressive Networks," Machine Learning at Berkeley, 16 August 2020. [Online]. Available: https://ml.berkeley.edu/blog/posts/AR_intro/. [Accessed 2 July 2021].

[8] J. Zhao, F. Huang, J. Lv, Y. Duan, Z. Qin, G. Li and G. Tian, "Do RNN and LSTM have Long Memory?," in *Thirty-seventh International Conference on Machine Learning*, Vienna, 2020.

[9] S. Bai, Z. Kolter and V. Koltun, "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling," *arXiv*, 2018.

[10] T. H. Trinh, A. M. Dai, M.-T. Luong and Q. V. Le, "Learning Longer-term Dependencies in RNNs with Auxiliary Losses," in *Thirty-fifth International Conference on Machine Learning*, Stockholm, 2018.

[11] N. Bitar, S. Muhammad and H. H. Refai, "Wireless technology identification using deep Convolutional Neural Networks," in *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, Montreal, 2017.

[12] E. Almazrouei, G. Gianini, N. Almoos and E. Damiani, "Robust Computationally-Efficient Wireless Emitter Classification Using Autoencoders and Convolutional Neural Networks," *Sensors*, vol. 21, no. 7, p. 2414, 2021.

[13] L. Bai, L. Yao, S. S. Kanhere, X. Wang and Z. Yang, "Automatic Device Classification from Network Traffic Streams of Internet of Things," in *43rd Conference on Local Computer Networks (LCN)*, Chicago, 2018.

[14] S. Rajendran, W. Meert, D. Giustiniano, V. Lenders and S. Pollin, "Deep Learning Models for Wireless Signal Classification With Distributed Low-Cost Spectrum Sensors," *IEEE Transactions on Cognitive Communications and Networking*, vol. 4, no. 3, pp. 433-445, 2018.

[15] X. Zhang, Y. Gao, Y. Yu and W. Li, "MUSIC ARTIST CLASSIFICATION WITH WAVENET CLASSIFIER FOR RAW WAVEFORM AUDIO DATA," *arXiv*, 2020.

[16] W. Balid, M. O. Al Kalaa, S. Rajab, H. Tafish and H. H. Refai, "Development of measurement techniques and tools for coexistence testing of wireless medical devices," in *2016 IEEE Wireless Communications and Networking Conference Workshops*, Doha, 2016.

[17] S. Ruder, "An overview of gradient descent optimization algorithms," 19 January 2016. [Online]. Available: https://ruder.io/optimizing-gradient-descent/index.html#adam. [Accessed 3 July 2021].

[18] R. Estes (rjalnev), "WIFINET [Source Code]," 2021. [Online]. Available: https://github.com/rjalnev/WIFINET.

Appendix A Random Samples for all Classes

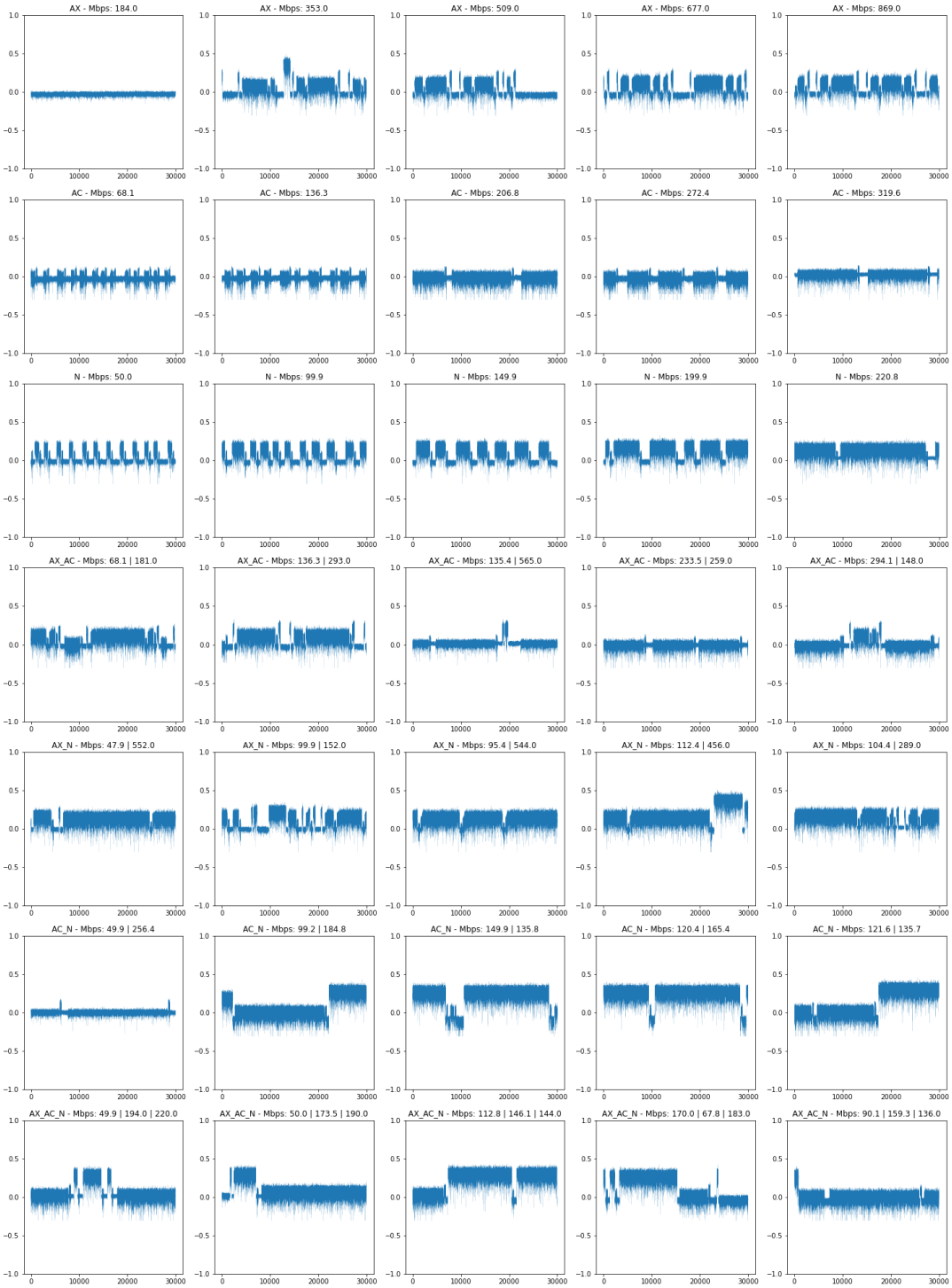

Figure A-1. Normalized random samples from all classes and throughputs.