Machine learning for wireless signal learning

Logan Smith
logan.smith.5@gmail.com

Machine learning for wireless signal learning

By

Logan Riggs Smith

Approved by:

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Computer Engineering
in the Department of Electrical and Computer Engineering

Mississippi State, Mississippi

April 2021

Name: Logan Riggs Smith

Date of Degree: April 30, 2021

Institution: Mississippi State University

Major Field: Computer Engineering

Major Professor: John E. Ball

Title of Study: Machine learning for wireless signal learning

Pages of Study: 53

Candidate for Degree of Master of Science

Wireless networks are vulnerable to adversarial devices by spoofing the digital identity of valid wireless devices, allowing unauthorized devices access to the network. Instead of validating devices based on their digital identity, it is possible to use their unique "physical fingerprint" caused by changes in the signal due to deviations in wireless hardware. In this thesis, the physical fingerprint was validated by performing classification with complex-valued neural networks (NN), achieving a high level of accuracy in the process. Additionally, zero-shot learning (ZSL) was implemented to learn discriminant features to separate legitimate from unauthorized devices using outlier detection and then further separate every unauthorized device into their own cluster. This approach allows 42% of unauthorized devices to be identified as unauthorized and correctly clustered

Key words: Zero-Shot Learning, Wireless Physical Fingerprint, Machine Learning

DEDICATION

To my mother, who took care of me after a serious eye injury, giving me time and space to consider my future. Without her hospitality, I would not have chosen to pursue a graduate degree.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Wireless networks identify devices by using a digital media access control (MAC) address, which is intended to be unique for every device. This allows wireless networks to choose which devices to allow access; however, this address can be easily mimicked, allowing unauthorized devices access to the network.

Wireless devices also have a "physical fingerprint" which is less easily mimicked. This unique identifier is caused by differences in the manufacturing of their hardware, leading to differences in each device's transmitted signal [16]. By using the physical fingerprint, MAC address mimicking is avoided since the MAC address is no longer used for identification.

A wireless network may have several devices that are intended to be authorized, with all other devices not being allowed on the network. When an authorized device tries to connect, the network should allow access. When an unauthorized device tries the same, the network should not allow access. Additionally, identifying and capturing inputs from unauthorized devices can be further utilized for other purposes, such as localization of the unauthorized device. Wireless devices can be separated and clustered using Zero-shot Learning (ZSL), where outlier detection is applied to these wireless signals to separate authorized and unauthorized devices. These unauthorized devices are then further clustered into their own set.

The contributions in this work to the physical device fingerprinting research area are:

1. Four real and complex neural networks (NN) are implemented for classification of wireless devices using their physical fingerprint. These networks showed state-of-the-art accuracy, implying that the physical fingerprint is indeed a valid classification input which is more robust than using the digital MAC address.
2. A ZSL method is implemented for the case of wireless physical fingerprints, which has not been attempted before in the literature.
3. A wireless signal dataset containing 9 classes with approximately 100 samples each.

Beyond the research here and in Chapters 3 and 4, this work has been extended by classifying long term evolution (LTE) signals and investigated the effect on classification for applying the short time Fourier transform with varying filter sizes to wireless signals. Additional code and experiments were written to perform localization of wireless devices for both real and simulated data.

CHAPTER II

BACKGROUND

Herein, an explanation of physical fingerprints is presented, alongside an overview of NNs used in this document and a review of the ZSL literature.

2.1 Physical Fingerprinting

When a wireless device is communicating to a receiver using a wireless protocol, MAC address identification occurs in the data link layer. The signal is originally obtained in the physical layer before being passed to the data link layer. By focusing on the signal obtained in the physical layer, the physical fingerprint of the device can be captured. This fingerprint is unique to that device due to differences in the analog circuitry of every device when they are manufactured [16], which is shown in differences in transmissions in the physical layer.

There have been several methods that have already utilized physical fingerprinting to increase robustness. The transient of the signal has been utilized [27], as well as the power [45], and the wavelet transform [28], and the magnitude and phase of each signal [48]. Dimensionality reduction of the data was implemented using principal component analysis [41] as well as autoencoders (AE) [51]. A comparison between using high-end and low-end receivers has also been employed [44].

2.2 WiFi

WiFi refers to a class of IEE 802.11 network protocols which allow wireless devices to connect to access points and the Internet. The most common radio bands used are 2.4 gigahertz (GHz) and 5 GHz. Data sent over these bands are formatted into 802.11 frames, using the MAC address when sending over a local area network (LAN).

2.3 Neural Networks

NNs are a collection of neurons that learn through training on a dataset. Given sufficient depth and appropriate non-linearities, NNs can learn any continuous function to any given desired precision [15]. This has allowed intellectual work to be offloaded onto NNs, saving time and even allowing superhuman performance. NNs have been shown to control a manned aircraft after simulated damage affecting maneuverability has occurred [25]. They have been used to analyze nuclear sites and their economic impact [36]. The medical field has made use of NNs for the purpose of sleep analysis [49] and decision making for potential intensive care patients [29].

Deep NNs (DNN) include a large class of NNs, including AEs, convolutional neural networks (CNN), and complex NNs which are presented in this work. The "deep" adjective means there are multiple hidden layers in the network.

Convolutional neural networks (CNN) apply convolution to the dataset (usually images) to detect high level features like edges and corners. It may also contain max pooling layers and dense layers at the end of the network. A popular CNN is Alexnet which achieved state-of-the-art accuracy in 2012 on the imagenet dataset [32].

AEs are NNs that first encodes data and then decodes it back into the original data. This allows the network to learn a compression of the data while optimizing for the minimal loss of information in the process. Many methods use the AEs compressed representation of the data for the purpose of dimensionality reduction and feature selection.

Complex NNs have complex-valued weights and activation functions. This greatly expands the possible representation space of the network parameters [52, 17].

Support vector machines (SVM) attempt to separate the inputs using a learned hyperplane. Many datasets cannot be linearly separated, though it is possible to avoid that issue by first projecting the data using a kernel before learning an optimal hyperplane. SVMs have been applied in many important applications, with one example being facial recognition [43]

2.4 Zero-Shot Learning

ZSL attempts to correctly classify unseen classes of data that were not seen in the training dataset. This is closely related to one-shot learning where one example of each unseen class is available during training. Typically in ZSL, extraneous semantic information is available for the unseen classes. For example, the unseen class of "zebra" contains the semantic labels of "black", "white", and "striped". As the network learns these semantic labels from other classes, it can then transfer that knowledge to recognize a zebra in the testing phase, even though it was not trained on any zebras in the training dataset. Mathematically, ZSL can be defined as follows:

Zero-Shot Learning: Denote $\mathcal{D}_{tr} = \{(x_i, y_i)\}$, for $i = 1, 2, \cdots, N_{tr}$, as the training data set consisting of N_{tr} labeled data samples. Assume that these N_{tr} training samples belong to a set of N_s seen classes: $C_{tr} = \{c_i^{tr}\}$, for $i = 1, 2, \cdots, N_s$. Also, denote $\mathcal{D}_{te} = \{x_i^{te}\}$, for $i = 1, 2, \cdots, N_{te}$, as a test data set, and each data sample x_i^{te} belongs to either one of the seen classes from C_{tr} or one of the unseen classes from $C_{te} = \{c_i^{te}\}$, for $i = 1, 2, \cdots, N_u$, where $C_{tr} \cap C_{te} = \varnothing$. Then, the goal of ZSL is to train a classifier that can predict class labels of all testing data samples in $\mathcal{D}_{te}$.

Although in cases such as wireless classification, meaningful semantic labels cannot be curated beforehand. It then falls to NNs to learn features that can differentiate unseen classes into reasonable clusters.

Once these features are learned, clustering is required to separate the dataset accordingly. Clustering is an unsupervised learning method which groups data points according to a similarity metric, such as their Euclidean distance and their density. K-means clustering assigns clusters based on euclidean distance from k randomly selected centroids. For many cases, it is unknown an optimal value to choose for k, though a useful heuristic is the "elbow" method where the variance of the data is plotted against the number of clusters [50]. A useful value for k is found in the "elbow" of the graph where most of the variance is explained for the least amount of clusters chosen, and greater values of k result in diminishing returns.

Density-based spatial clustering of applications with noise (DBSCAN) is another clustering algorithm, though this method relies on a closeness parameter ϵ and density (minPts) as opposed to a pre-specified value k for the number of clusters [20]. For every point, if it is has minPts neighbors within ϵ-distance, it is considered part of that cluster with those neighbors. Points without a sufficient amount of neighbors are considered noise.

CHAPTER III

CLASSIFYING WIFI PHYSICAL FINGERPRINTS USING COMPLEX DEEP LEARNING

3.1 Abstract

Wireless communication is susceptible to security breaches by adversarial actors mimicking Media Access Controller (MAC) addresses of currently-connected devices. Classifying devices by their "physical fingerprint" can help to prevent this problem since the fingerprint is unique for each device and independent of the MAC address. Previous techniques have mapped the WiFi signal to real values and used classification methods that support solely real-valued inputs. In this paper, four new deep neural networks (NNs) are implemented for classifying WiFi physical fingerprints: a real-valued deep NN, a corresponding complex-valued deep NN, a real-valued deep CNN, and the corresponding complex-valued deep convolutional NN (CNN). Results show state-of-the-art performance against a dataset of nine WiFi devices.

3.2 Introduction

In order to classify the "physical fingerprint", only the preamble of each wireless signal is used. The preamble for the 802.11a/g wireless protocol can be broken down into three subcomponents: the Short Training Field (STF), the Long Training Field (LTF), and the Signal (SIG). This provides information regarding synchronization and data length (or how to decode and how much to decode). Importantly, information on the MAC address is absent, making it invariant to MAC-address spoofing. Sampling at 20 MHz with a software-defined radio (SDR), the preamble is located in the first 400 samples of each signal as shown in Figure 3.1.

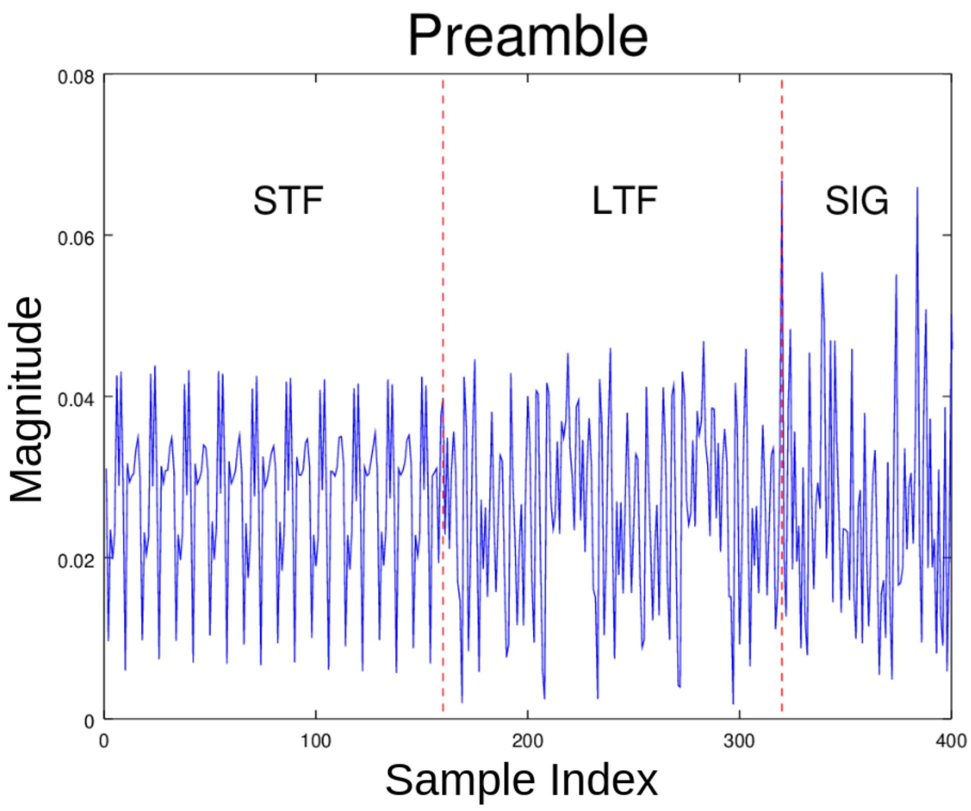


Figure 3.1: Preamble of a wireless signal in the 802.11a/g protocol. STF=Short Training Field, LTF=Long Training Field, and SIG=Signal.

Herein, real-valued and complex-valued deep neural networks (DNNs) and deep convolutional neural networks (CNNs) are investigated to classify WiFi preamble signals. The contributions in this work to the physical device fingerprinting research area are:

1. A high-performance real-valued DNN has been developed and validated.
2. A complex-valued DNN that mirrors the architecture of the corresponding real-valued DNN has been developed and validated.
3. A high-performance real-valued deep CNN has been developed and validated.

4. A complex-valued deep CNN that mirrors the architecture of the corresponding real-valued deep CNN has been developed and validated.
5. All four networks show performance that is at or above state-of-the-art methods for physical device fingerprinting.

This section will discuss some basics of NNs and support vector machines (SVMs) and discuss previous methods in device fingerprinting analysis.

3.3 Device Fingerprinting

Device fingerprinting relies on hardware devices having a unique digital transmission signature [30, 4, 21, 10, 46]. Radio frequency (RF) fingerprinting is challenging due to multipath and channel fading effects, component aging and temperature effects on electronics [46, 21], as well as hardware related imperfections and differences between individual units that cause modulation errors and artifacts [9]. There are several survey papers on this topic as device fingerprinting is an active area of interest [37, 10, 21, 7, 57, 22, 59, 54, 39, 23, 2, 5].

Over the past several years, a number of methods have been proposed for device fingerprinting [4, 21]. A common approach is to rely on the usage of RF technology. For example, many RF identification systems (RFIDs) are used in a wide variety of applications due to their resistance to multipath effects and the complexities of indoor systems [21]. One method, in particular, takes advantage of RF technology by using Bayesian change detectors to find RF fingerprints [53]. However, performance is dependent on specific equipment, and the corresponding energy consumption may be prohibitive in certain settings. Chen et al. utilized infinite hidden Markov random fields for device fingerprinting [12]. Lanze et al. used RF clock skew to analyze RF fingerprints [33]. Some proposed solutions utilize the transient signal for analysis. Klein et al. performed a sensitivity study using two different techniques [27]. Rehman et al. examined the energy envelope of signal transients for RF fingerprint extraction [45]. Klein et al. utilized wavelet transforms to analyze RF fingerprints [28].

A mobile approach to fingerprinting has also been investigated. Khullar and Dong opted for a client-server approach using cell phones to scan and extract data from the WiFi access points. Afterward, the collected data is processed through SVMs, in order to train and test their system [24]. They found that adding temporal features greatly enhanced location prediction accuracies. Suski et al. put forth a digital fingerprint classifier system. The system first estimated the instantaneous amplitude, phase for each time sample and then estimated the transient start-up location [48]. Based on the transient location, they extracted statistical and parametric features from the signal and applied Fisher's linear discriminant analysis and spectral correlation. They achieved about 83% and 93% overall localization accuracies using spectral correlation and multiple discriminant accuracy, respectively, with three classes and a 6 dB signal to noise ratio. Padilla et al. utilized principal components analysis and partial least squares regression to classify digital fingerprints. They classified ten different devices and the best results were about 94% overall accuracy [41].

Brik et al.'s Passive RAdiometric Device Identification System (PARADIS) approached fingerprinting by specifically identifying hardware modulation differences such as IQ encoder errors, self-interference, frequency errors and amplitude clipping [9]. They tested their system on a relatively large set of data (138 different devices). Their approach works in the modulation domain by estimating the IQ origin offsets , frequency differences, Sync correlation behavior, magnitude error and phase error between ideal and observed data over multiple symbols in a frame. An Agilent 89641S vector signal analyzer was used to capture the data. They achieved excellent performance of about 99.6% overall accuracy. However, their approach requires sensitive and very costly equipment (\$100K range) to estimate the error data and many frames to perform analysis. Rehman et al. has examined low-end and high-end receivers, and concludes that low-end receivers (SDRs) require a higher SNR to achieve good performance compared to high-end receivers [44]. High-end receivers definitely have a performance advantage over low-end SDRs.

Many NN are called deep networks or DNNs, because they have many layers. DNNs have been used successfully in many areas including image processing, remote sensing, etc. [6]. A special class of DNNs is a CNN. A CNN is loosely based on the human and primate visual system. A typical CNN operates on imagery and usually employs multiple groups 2D masks followed by pooling and some nonlinear activation such as a rectified linear unit (relu). Usually, at the tail end of the network, there are some fully connected (Dense) layers followed by a softmax layer for classification.

Recently, DNNs and CNNs have been applied for RF fingerprint analysis. For example, using 2D radio maps as inputs, Jang and Hong developed a system using a CNN for indoor localization [23]. An attractive feature of CNNs is that they have fewer parameters than DNNs; consequently, using a CNN allows for improved execution times and offers more sensitivity to power fluctuations caused by multipath effects. They used computer access points (APs) to collect data and analyzed the received signal strength indicator (RSSI) values. They transformed the RSSI vector by adding some padding data and reshaping it into an image then applied a deep CNN. Their goal was to estimate the building ID and floor in the building, and they achieved about 95% accuracy. Nowicki and Wietrzykowski proposed using DNNs to facilitate learning from the data rather than tedious hand-tuning for fingerprinting [40]. Specifically, they used a diabolo-shaped stacked autoencoder (AE) for dimensionality reduction. They also utilized RSSI information and analyzed the UJIIndoorLoc dataset [51], which contains 21,048 WiFi scans recorded by 25 Android devices. The AE inputs 520 RSSI values in the dataset. Their system examined results on building and floor localization, and achieved about 91% accuracy on the test subset. Merchant et al. used three deep learning networks with exponential activation units (ELUs) [14] to recognize IEEE 802.15.4 devices with about 92.3% overall accuracy [38].

3.4 Neural Networks

NNs are loosely based on human neurons. Each fully-connected neuron accepts inputs and calculates a dot product of the inputs and adds a bias term. The neuron output is the processed with a linear or non-linear function called the activation function. A standard DNN is composed of many interconnected layers of neurons. A process called backpropagation is used to adjust the neuron weights and bias terms. This process is called training the network, and there exist several training methods in the literature; herein, the Adam (adaptive moment estimation) method [26] is used for network training.

Given sufficient depth and appropriate non-linearities, NNs can learn any continuous function to any given desired precision [15]. This property makes NNs an attractive choice for machine learning and signal classification. However, NNs must also contend with the problem of overtraining. To mitigate this, the network size and complexity can be adjusted, and dropout layers can be utilized. Dropout layers randomly set some specified fraction of these inputs to zero during training. In most cases, this forces the network to generalize better. During the inference phase (giving the network testing data), all of the signals are passed through the dropout layer.

3.4.1 Complex NN

Complex NNs are less common in the literature and it seems likely that there exists only one implemented example [52, 17]. One may expect they are uncommon because most deep-learning networks operate on real-valued data such as images or time sequences. Radar data and receiver data in communication systems are instances where the data is inherently complex, and these systems are beginning to implement deep learning, but even then, oftentimes these systems transform the complex data and utilize real-valued networks. Moreover, there is a very significant amount of software available for deep learning with real-valued network, such as Tensorflow, Keras, PyTorch, Matlab's deep learning toolbox, etc. These heavily-used systems mostly provide support for real-valued networks, so it is natural that their might be a scarcity of choices for complex-valued networks.

Unfortunately, this implementation is a poor fit for the application. While this complex NN can process complex data internally, its layers accept only real data as inputs. Moreover, a critical function, ComplexConv1D, does not support the same input shape as TF's one-dimensional convolution layer, *Conv1D*. In TF, the input tensor is formatted as `(None, data_vector_size, 1)`, where `None` indicates that TF can select the batch size. However, ComplexConv1D takes inputs `(None, data_vector_size)`. Consequently, TF Conv1D was used as the first layer in the complex CNN.

A typical real-valued NN might use a rectified linear unit (relu) operator to provide a non-linearity, where a relu is defined as

$$relu\left(x\right) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}, \tag{3.1}$$

where x is a real-valued variable. However, the real-valued relu is not appropriate for a complex-valued input. One suggested replacement is called the "modrelu" [56]. The modrelu utilized for a complex value z with real-valued parameter b is

$$modrelu\left(z\right) = relu\left(|z| + b\right)\frac{z}{|z|}. \tag{3.2}$$

Herein, the modrelu is modified slightly to prevent division by zero. The Keras code is listed in Appendix A and the modrelu used herein (with $\varepsilon = 1.0 \times 10^{-12}$), which is a small value utilized to prevent dividing by zero. Other values could have been chosen as well. The modified modrelu as shown below in eq. (3.3)

$$modrelu\left(z\right) = relu\left(|z| + \varepsilon + b\right)\frac{z}{|z| + \varepsilon}. \tag{3.3}$$

The complex networks was difficult to train using the suggested modrelu parameter of $b = -0.5$, so instead $b = 0.01$ was used.

3.5 Support Vector Machines

A SVM is a shallow-learning methods that uses an objective function to find an optimal separating hyperplane, either in the input data space or in the kernel space, that best separates the data samples. A linear SVM learns an optimal separating hyperplane from the training data, and this decision boundary is then applied to the testing data. If the data are slightly non-linearly separable, slack variables can be used. However, if the data are highly nonlinear, then the linear SVM will provide sub-optimal results. The second approach is to utilize kernel machines, which project the data into a nonlinear space (when designed correctly this space will make the data nearly linearly separable) and then use a linear SVM in the projected space. The non-linear SVM is called a kernel SVM.

Herein, the radial basis function (RBF) kernel, which has one parameter that controls the RBF variance, was utilized. The SVM itself has a cost parameter, which is the penalty incurred during optimization for classification errors. The rfb kernel SVM thus has two hyperparameters, the cost C and the RBF kernel size. The RBF kernel with two input vectors $\mathbf{x}$ and $\mathbf{y}$ is

$$K(\mathbf{x}, \mathbf{y}) = exp\left(\frac{-\|\mathbf{x} - \mathbf{y}\|^2}{2\sigma^2}\right). \tag{3.4}$$

The scikit SVM [42], which is a wrapper around libsvm [11], was used.

3.6 Methods

3.6.1 Development Environment

Herein, Keras [13] and Tensorflow (TF) [1] are used for implementing and testing the various NN architectures. Anaconda [3] was utilized to install the required packages as shown in Appendix A.10.

TF 2 was utilized with graphical processing unit (GPU) support for faster training. TF GPU is not required, the CPU version is fine as well. Instructions for installing TF GPU can be found here: https://towardsdatascience.com/tensorflow-gpu-installation-made-easy-use-conda-instead-of-pip-52e5249374bc.

3.6.2 Datasets

To collect the signals for this dataset, IEEE 802.11a/g WiFi traffic were captured from nine different devices: Five WiFi routers, a Google Pixel 2XL, a Samsung Galaxy J2 Prime, an Oppo R11, and a Blackberry Curve 3G 9300 phone. This variety of transmitters was intended to prove the efficacy of the radio fingerprinting process. A USRP B210 SDR was used to collect the complex data using GNURadio [19]. The USRP B210 offers many desirable features for this experiment, such as a frequency range of 70 MHz - 6GHz, instantaneous bandwidth of 200 KHz - 56 MHz, sampling rates up to 61.44 MS/s, and receiver front-end gains of up to 76dB. Ultimately, traffic was captured at a 20 MHz sampling rate for approximately two minutes.

Building off of Bloessl's WiFi receiver [8], an autocorrelation function detects the STF sequence of the WiFi frame based on its periodicity. Then, the receiver saves a copy of the raw frame with a generated counter and attaches that same counter to the original. The original continues being processed until the MAC address can be extracted and saved to file with the counter.

There are cases where the MAC address cannot be extracted; for example, due to a corrupted frame. When collecting data, any frames where such an error occurs were thrown out. Additionally, frames containing an all-zero MAC address were removed. However, there were still frames where the noise was constant and cyclical over the whole frame, as opposed to just the STF sequence of a normal frame. This is one reason to explain why the autocorrelation function categorized it as a normal WiFi frame. Thus, any frames whose maximum correlation was not 1 higher than the average autocorrelation were filtered out.

When recording data, the first sample was flagged if its amplitude was greater than 0.05, keeping the next 400 samples which represent the preamble. The amplitude constant was set empirically based on the receiver noise characteristics. Signals that had a preamble zeroed-out midway through were discarded. Signals from wireless devices that provided less than 90 instances each were disposed. The remaining signals were split into real and imaginary components, leading to a signal with 800 real samples representing 400 complex samples.

Additionally, approximately 0.1% of signals were not wireless preambles, but still passed the autocorrelation threshold by coincidentally having the same periodicity of the preamble's STF. These were thrown out manually. This dataset contained preambles of nine wireless devices with about 100 instances each.

3.6.3 Data Preprocessing

The data was collected using an SDR, and verified manually. This data is IQ data. The preamble consists of 400 complex samples. Most NNs are not designed to ingest complex data. As such, there are several options available to the network designer. One option is to encode the complex data as in eq. 3.5 by interleaving the real and the imaginary for each complex sample.

$$\mathbf{x}^C = \left[RE\left\{x_0\right\}, IM\left\{x_o\right\}, RE\left\{x_1\right\}, IM\left\{x_1\right\}, \cdots, RE\left\{x_{N-1}\right\}, IM\left\{x_{N-1}\right\}\right]^T. \tag{3.5}$$

Here, $RE\{x\}$ and $IM\{x\}$ denote the real and imaginary portions of x, respectively, the superscript T denotes a transpose operator, and N is the data vector size (400 complex samples). Herein, the real imaginary encoding method shown in eq. 3.5 is utilized.

NNs can be sensitive to the data ranges. To normalize the data, the maximum values of the real and complex data were used. The data was normalized by dividing all samples by the maximum amplitude plus 0.1 for overhead. The normalized data was randomly shuffled and split into three disjoint datasets: training (60 %), validation (10 %) and testing (30 %). The data splitting kept the same approximate class ratios in each split. In the experiments, the network was run ten times, each time preprocessing, randomizing, and splitting the data.

Finally, since there were not enough training samples for the network sizes, data augmentation was used. For each data vector, ten new samples were created by taking the original vector and adding a random constant phase shift to every interleaved sample prior to processing by eq. 3.5. That is, a single phase offset was applied to the complex data. The rationale is the phase of the data is inherently unknown, so this accounts for the uncertainty in this parameter while also providing augmented data samples for the system.

3.6.4 Network Architectures

Several network architectures are investigated for their ability to distinguish different phone signatures. Herein, four NN architectures are examined: (1) a deep, real-valued NN, (2) a deep, complex-valued NN, (3) a deep, real-valued CNN, and (4) a deep, complex-valued CNN.

3.6.4.1 Complex-valued and Real-valued DNNs

Herein, two DNNs are analyzed, a real-valued DNN (DNN-R1) and a complex-valued DNN (DNN-C1). The network architectures for the DNN-R1 and DNN-C1 networks are shown in Tables 3.1 and 3.2, respectively. Keras codes for these two networks is given in Appendix A. In addition, to provide comparison methods to previous methods in the literature, a kernel SVM was utilized to provide a baseline. Dense is a TF fully-connected layer. The rate for a dropout layer is the fraction of entries that are randomly zeroed during each training minibatch. The number of classes is C. The Abs layer calculates the absolute value of the complex input and casts to TF *float32* datatype.

Table 3.1: Architecture of DNN-R1. C is the number of classes.

Layer	Size	Options
Input	(None, 800)	
Dense	800	relu activation
Dense	200	relu activation
Dense	50	relu activation
Dropout		0.50 rate
Dense	C	softmax

Table 3.2: Architecture of DNN-C1. C is the number of classes.

Layer	Size	Options
Input	(None, 800)	
ComplexDense	800	modrelu activation
ComplexDense	200	modrelu activation
ComplexDense	50	modrelu activation
Abs		
Dropout		0.60 rate
Dense	C	softmax

3.6.4.2 Complex-valued and Real-valued CNNs

The network architectures of the complex-valued CNNs are shown in Tables 3.3 and 3.4. The CNN network uses 32 filters of size 12 followed by 2×2 max pooling and 64 filters of size 3, followed by two 2×2 max pooling layers. A dropout layer with loss rate 0.5 is used to mitigate overtraining. The data are flattened and then passed through two 32-neuron fully connected (dense) layers and a final dropout layer before the softmax activation layer. The network CNN-C1 is the complex equivalent of CNN-R1. The main differences are the second set of convolution layers are complex, and the two fully connected layers after the flattening layer are also complex. The first layer is a regular Conv1D, since the ComplexConv1D implementation was not able to accept inputs compatible with the rest of the network. Also, a complex dense layer of size K has $2K$

components (one each for real and complex), so the final layer would have $2C$ outputs, which did not match the dataset with nine classes. For that reason, the last softmax layer is a real-valued fully connected layer.

Table 3.3: Architecture of CNN-R1. C is the number of classes.

Layer	**Size**	**Options**
Input	(None, 800, 1)	
Conv1D	32	kernel size 12 relu activation same padding strides of 2
MaxPool1D	2	
Conv1D	64	kernel size 3 relu activation valid padding strides of 2
MaxPool1D	2	
MaxPool1D	2	
Dropout		rate 0.5
Flatten		
Dense	32	relu activation
Dense	32	relu activation
Dropout		rate 0.5
Dense	C	softmax activation

3.6.5 Network Sizes

One important consideration for any DNN or CNN is the total number of trainable parameters. This affects the network performance as well as sets requirements on the minimum size of the training set. Table 3.5 shows the overall number of parameters for each network. From the table, there are a few more layers in DNN-C1 due to the conversion to absolute value. Also, in terms of trainable parameters, CNN-C1 is about double the size of CNN-R1, which is expected, since the ComplexConv1D will have double parameters to handle real and imaginary numbers. The CNN networks are much smaller and perform relatively well.

3.6.6 Training parameters

Table 3.6 lists the training parameters for each network. The term "None" in the input size is the encoding TF uses to allow batches to be any size. The number of training epochs and batch sizes were adjusted to provide strong performance results while minimizing overtraining. It is hard to directly compare to other classification results in the literature since the datasets are different. The best results in the literature are currently PARADIS [9], but if you consider the very expensive hardware required for their system versus the an inexpensive SDR solution, state-of-the-art performance was achieved in device fingerprinting classification.

3.7 Results and Discussion

In this study, four NN's were compared: A real-valued DNN (DNN-R1) and it's complex-valued counterpart (DNN-C1), a real-valued CNN (CNN-R1) and it's complex-valued counterpart (CNN-C1), and a RBF-SVM. To optimize the SVM parameters, a parameter sweep was performed across the penalty parameter and the RBF size. Fig. 3.2 shows the results of the parameter sweep for training (unaugmented dataset). Based on these results, the RBF standard deviation parameter was selected as 4.096 and the cost as 38.0. The overall results are tabulated in Table 3.7. The results show the overall accuracies for the raw complex data and the FFT-processed complex data. The best results were obtained with DNN-C1 on the FFT-processed data, achieving 98.81% overall accuracy. Network DNN-R1 was a close second with 98.14% overall accuracy on the raw complex data. The SVM had the next best at 96.88% overall accuracy, followed by CNN-R1, with 96.06% overall accuracy.

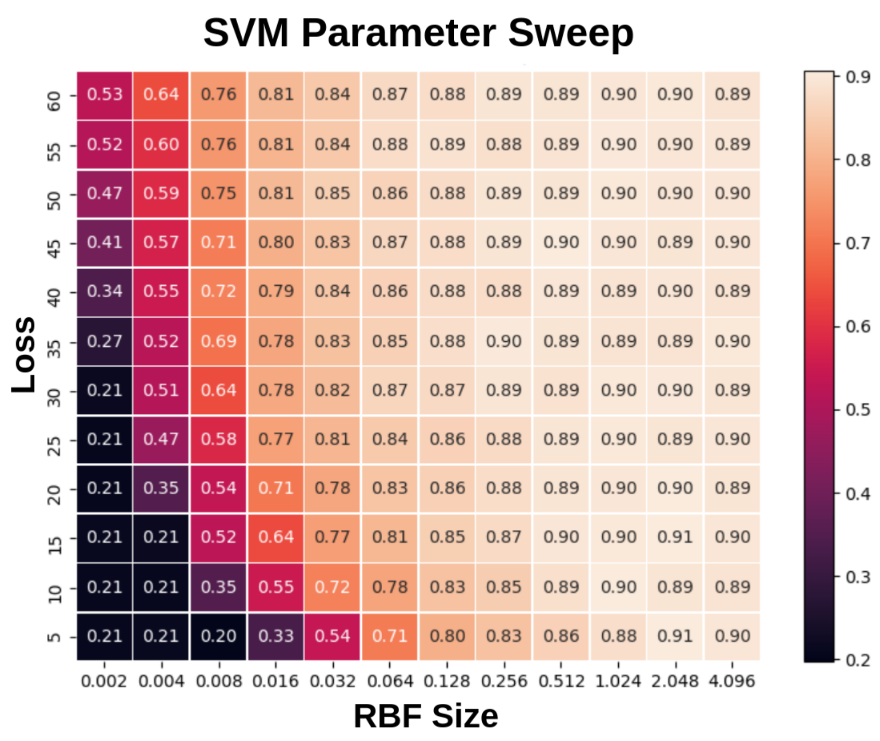


Figure 3.2: SVM parameter sweep over the kernel size and loss parameters. Individual values in each square represent the percent accuracy achieved by the SVM averaged over ten runs. Lighter colors mean better results. Best viewed in color.

From these results, it is seen that DNN-C1, the complex-valued version of DNN-R1, slightly outperformed real-valued network for the FFT-processed data. Furthermore, in all cases except DNN-R1 (which was very close), the FFT-processed data gave better results. This was expected, since the time-domain data can have phase shifts. A delay in the time domain would just induce a phase shift in the FFT domain.

Table 3.8 shows the testing average confusion matrix for CNN-C1 over ten runs. Classes 0 - 4 are routers, class 5 is a Blackberry Curve, 6 is an Oppo R11, 7 is a Google Pixel 2XL and 8 is a Samsung Galaxy J2 Prime. The most confusion was found for class 4, which the classifier confused between the other routers. There was a small amount of confusion between classes 5, 6, and 7.

3.8 Conclusions and Future Work

In this paper, four networks were created that provide state-of-the-art performance for device ID fingerprinting based on the 400-sample preamble. Very high classification accuracies were obtained for all of the proposed networks, especially DNN-R1. For data captured via SDR, state-of-the-art RF classification performance was achieved. It was also demonstrated that complex NN are feasible alternatives for complex-valued signal processing. Finally, in almost all cases, there were performance gains by pre-processing the signals with a FFT.

Future work includes testing on a larger set of devices, implementing a complex NN that natively handles complex inputs, testing different network architectures and optimizing via a genetic algorithm. Furthermore, more training samples can benefit larger networks, so it is desirable to perform a large-scale data collection. Finally, it would be beneficial to investigate modifying the complex NN 1D convolution to be a drop-in replacement for a real-valued TF 1D convolution.

Table 3.4: Architecture of CNN-C1. C is the number of classes.

Layer	Size	Options
Input	(None, 800, 1)	
Conv1D	32	kernel size 12 modrelu activation same padding strides of 2
MaxPool1D	2	
ComplexConv1D	64	kernel size 3 modrelu activation valid padding strides of 2 complex_independent kernel initializer
MaxPool1D	2	
MaxPool1D	2	
Dropout		rate 0.5
Flatten		
ComplexDense	32	modrelu activation
ComplexDense	32	modrelu activation
Dropout		rate 0.5
Dense	C	softmax activation

Table 3.5: Network number of layers and total number of trainable parameters.

Network	Number of Layers	Total Trainable Parameters
DNN-R1	5	811,509
DNN-C1	7	982,559
CNN-R1	11	57,161
CNN-C1	11	107,753

Table 3.6: Training parameters and settings for the DNN and CNN architectures.

Network / Parameter	DNN-R1	DNN-C1	CNN-R1	CNN-C1
Optimizer	Adam	Adam	Adam	Adam
Learning rate	2.0e-3	2.0e-3	2.0e-3	2.0e-3
Data vector size (N)	800	800	800	800
Input Size	(None,N)	(None,N)	(None,N,1)	(None,N,1)
Epochs	6	67	57	57
Batch size	100	500	725	725

Table 3.7: Experiment results showing overall test accuracies in percent. Best results for each method are shown in bold text.

Method	**Complex Data (No FFT)**	**Complex Data (FFT Processed)**
Real-valued NN (DNN-R1)	**98.14**	98.01
Complex-valued NN (DNN-C1)	96.09	**98.81**
Real-valued CNN (CNN-R1)	95.20	**96.06**
Complex-valued CNN (CNN-C1)	92.63	**93.95**
SVM	95.31	**96.88**

Table 3.8: Overall average confusion matrix for the testing data for DNN-C1 over ten runs. The true values are in columns, and the network classifier outputs are in rows. The bold diagonal elements are correctly classified.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
0	**1333.3**	0.0	0.0	2.4	7.0	0.0	9.6	0.0	0.0
1	1.1	**1363.0**	3.9	0.0	7.4	5.6	0.0	0.0	6.3
2	1.1	0.0	**1375.1**	0.0	0.0	0.0	0.0	4.6	5.4
3	0.0	0.0	0.0	**1376.9**	0.0	6.4	0.0	8.8	0.0
4	13.2	3.1	7.0	6.7	**1266.1**	4.1	0.0	0.0	0.90
5	0.0	1.9	0.0	0.0	2.1	**1065.5**	0.0	0.0	0.0
6	9.3	0.0	0.0	0.0	0.0	5.2	**901.4**	5.1	0.0
7	0.0	0.0	0.0	0.0	0.4	1.2	0.0	**1069.5**	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**1029.4**

CHAPTER IV

IDENTIFYING UNLABELED WIFI DEVICES WITH ZERO-SHOT LEARNING

4.1 Abstract

In wireless networks, MAC-address spoofing is a common attack that allows an adversary to gain access to the system. To circumvent this threat, previous work has focused on classifying wireless signals using a “physical fingerprint”, i.e., changes to the signal caused by physical differences in the individual wireless chips. Instead of relying on MAC addresses for admission control, fingerprinting allows devices to be classified and then granted access. In many network settings, the activity of legitimate devices—those devices that should be granted access—may be dynamic over time. Consequently, when faced with a device that comes online, a robust fingerprinting scheme must quickly identify the device as legitimate using the pre-existing classification, and meanwhile identify and group those unauthorized devices based on their signals.

This paper presents a two-stage Zero-Shot Learning (ZSL) approach to classify a received signal originating from either a legitimate or unauthorized device. In particular, during the training stage, a classifier is trained for classifying legitimate devices. The classifier learns discriminative features and the outlier detector uses these features to classify whether a new signature is an outlier. Then, during the testing stage, an online clustering method is applied for grouping those identified unauthorized devices. This approach allows 42% of unauthorized devices to be identified as unauthorized and correctly clustered.

4.2 Introduction

There is a need to separate legitimate from unauthorized devices using a mix of both supervised and unsupervised learning using ZSL. Once separated, the system needs to further isolate each unauthorized device into its own cluster. This will allow specific information on each unauthorized device to be available for further purposes such as localization of individual devices and method of attack.

In the online setting, there is only access to legitimate devices beforehand for training; therefore, labels exist for legitimate devices, but not for unauthorized ones. It follows that supervised model cannot be solely used; however, to exclusively use an unsupervised method would be throwing away useful information about available legitimate devices. This motivates the use of ZSL, which is a form of semi-supervised learning where it is possible to leverage information regarding observed classes (legitimate devices) in order to better separate unseen classes (unauthorized devices) into their own clusters. Figure 4.1 illustrates how generalized zero-shot learning applies to this specific use case.

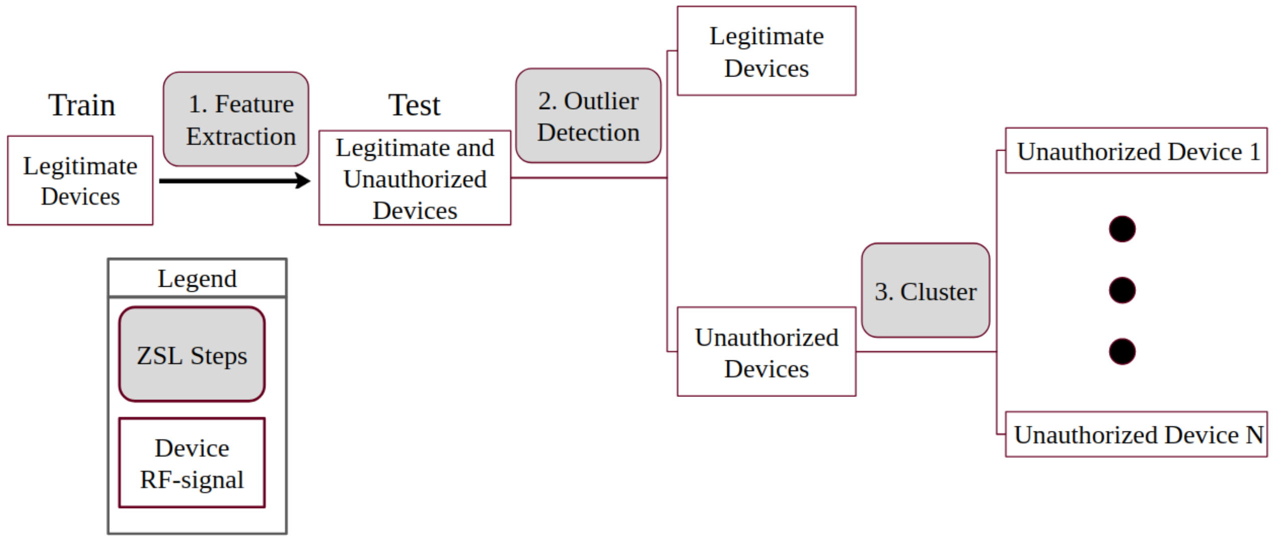


Figure 4.1: Overview of generalized zero-shot learning for this test case.

It is common to use additional semantic information to aid classification of unknown classes (such as “black, white, stripes” to classify “zebra”). Unfortunately, in this case, there is not meaningful semantic descriptions for wireless devices. The general form of ZSL without using additional semantic information is as follows:

1. Feature extraction: learning discriminative features helps identify outliers and cluster unseen classes.
2. Outlier detection: outliers will represent unknown/unauthorized devices.
3. Clustering of outliers: outliers will consist mostly of unknown/unauthorized devices, which there are no labels for; therefore, there is a need to separate each device into it’s own cluster.

Here, the main contributions are (i) identifying features of wireless data that lend themselves to outlier detection, and (ii) finding a different set of features more suited for clustering unseen devices.

4.3 Background and Related Work

The problem of identifying unknown devices can be formulated as a ZSL problem where the seen classes (i.e., legitimate devices) in the training data set and the unseen classes (i.e., unauthorized devices) to be classified in the test data set are disjoint. Mathematically, the ZSL can be defined as follows:

Zero-Shot Learning: Denote $\mathcal{D}_{tr} = \{(x_i, y_i)\}$, for $i = 1, 2, \cdots, N_{tr}$, as the training data set consisting of N_{tr} labeled data samples. Assume that these N_{tr} training samples belong to a set of N_s seen classes: $C_{tr} = \{c_i^{tr}\}$, for $i = 1, 2, \cdots, N_s$. Also, denote $\mathcal{D}_{te} = \{x_i^{te}\}$, for $i = 1, 2, \cdots, N_{te}$, as a test data set, and each data sample x_i^{te} belongs to either one of the seen classes from C_{tr} or one of the unseen classes from $C_{te} = \{c_i^{te}\}$, for $i = 1, 2, \cdots, N_u$, where $C_{tr} \cap C_{te} = \emptyset$. Then, the goal of ZSL is to train a classifier that can predict class labels of all testing data samples in $\mathcal{D}_{te}$.

Unlike many traditional machine learning methods which focus on classification problems where all classes are available in the training data set, ZSL requires incrementally learning and identifying new unseen classes during the testing stage. Many ZSL algorithms have been developed for the past decade with a wide range of applications in computer vision, robotics, and natural language processing. The rest of this section presents a brief review of related works in zero-shot learning, while interested readers can be referred to a more comprehensive survey of the literature, such as Wang et al.[55].

A ZSL algorithm typically falls into the following two categories: instance-based methods and classifier-based methods. While the former aims to leverage labeled instances for learning classifiers of unseen classes, the latter focuses on the learning of classifiers directly upon instances from both seen and unseen classes. Existing classifier-based methods usually apply a one-vs-rest scheme to build classifiers for each class, and three types of approaches have been developed: correspondence methods [18, 34, 35], relationship methods [31, 58, 60], and combination methods [47]. The instance-based approaches focus on the building of labeled instances for the unseen classes through projection, instance-borrowing, and synthesizing. Note that most existing zero-shot learning methods are built by exploring the relations of seen and unseen classes in their attributes and semantics. While these methods perform well in many image-analysis tasks, where meaningful semantics exist to describe instances, they would fail in this study of RF signal analysis to identify unseen devices whose fingerprints are difficult to describe with explicit attributes.

This paper presents a two-stage approach to address this problem. During the training stage, a classifier is first trained to learn discriminative features and recognize those legitimate devices. During the online testing stage, with the learned features, an outlier detector is used to identify all test instances belonging to the unseen classes, and apply a clustering algorithm which groups those unseen class instances, i.e., outliers, in terms of their similarity in the feature space.

Density-based spatial clustering of applications with noise (DBSCAN) is one such clustering algorithm. This method uses two parameters: ϵ which defines how close two datapoints need to be and minPts which specifies the number of neighbors required [20]. For every point, if it is has minPts neighbors within ϵ-distance, it is considered part of that cluster with those neighbors. Points without a sufficient amount of neighbors are considered noise.

4.4 Methods

4.4.1 Dataset

To collect the dataset for this project, USRP B210 SDR and GNURadio were used. Building off of Bastian Bloessl's WiFi receiver [8], an autocorrelation function detects the STF sequence of the WiFi frame based on its periodicity.

The magnitude of each complex signal was recorded, and due to noise, the first sample of each signal that had an amplitude greater than 0.05 was flagged, keeping the next 400 samples which represent the preamble. The threshold was empirically determined based on the noise levels of the SDR. Signals that had a preamble zeroed-out midway through were discarded. Signals of wireless devices that contained less than 90 instances each were thrown out, as having less would cause insufficient training. The remaining signals were then scaled individually to unit norm using Sklearn's normalize function [42].

Additionally, approximately 0.1% of signals were not wireless preambles, but still passed the autocorrelation threshold by coincidentally having the same periodicity of the preamble's STF. Such signals were discarded manually. Ultimately, the dataset contained preambles of nine wireless devices with approximately 100 instances each.

4.4.2 Models

Four neural network models were implemented for learning discriminative features in an either supervised or unsupervised manner: convolutional neural network (CNN), multi-layer perceptron (MLP), Autoencoder (AE), and Autoencoder with transductance (AE-transductance). These were implemented using custom networks from the keras library [13] in Python with layer-specific code available in Appendix A.7-9. All learning rates were set to their default values.

Convolutional Neural Net. For the CNN, a one-dimensional convolutional layer was chosen. Two-dimensional CNN's are typical for image inputs in order to learn image-related features such as edges and corners. On the other hand, a one-dimensional CNN is able to learn temporal features in time-series data which is suited for the preamble signals. For both the CNN and the MLP below, adagrad and categorical crossentropy were used for the optimizer and loss function, respectively.

Multi-layer Perceptron. As a base comparison against the CNN, an MLP was implemented containing three Dense layers with 50 units separated by dropout layers with dropout parameter set to 0.3. This means 30% of input units from the previous layer will be randomly zeroed during training to help prevent overfitting. During testing, all layers are passed through the dropout layer.

Autoencoder. An eight-layer diabolo-shaped AE was implemented with 200, 100, 40, and 13 neurons in the encoder layers, and vice-versa for the decoder layers. The embedded layer of size 13 was empirically determined to perform better than marginally smaller and greater values. Particularly, "tanh" was used for the last layer's activation function, mean absolute error for the loss function, and Adamax for the optimizer as these scored highest for reconstruction error and DBSCAN [20] clustering accuracy.

Autoencoder with Transductance An AE-transductance model was also implemented. All parameters were similar to the regular AE except it was trained on all RF-signals as opposed to only the legitimate devices. The motivation is to learn features of the unauthorized devices for which no labels exist.

4.4.3 Outlier Detection

In all cases, the legitimate devices encompassed five classes, leaving four unauthorized device classes. All experiments were averaged over ten runs using a different selection of wireless devices split between legitimate and unauthorized. Each of the networks were trained on 70% of the legitimate devices, and the remaining 30% and all unauthorized devices were encoded in each network for testing. Encoding for the CNN and MLP is defined as the k-layer output of each network. Encoding for the AE is the middle layer output. These encoded signals were then passed to a local outlier detector (LOD).

LOD is an unsupervised outlier detection method that determines outliers by their distance from clusters based on the density of the cluster, i.e., a datapoint is more likely to be labeled as an outlier if it is far from a dense cluster as opposed to a sparse cluster. In order to leverage this unsupervised method, the LOD was ran with all encoded training data and one of the encoded legitimate or unauthorized devices. If the training data is classified as an outlier, that information is ignored since there will be training data with labels in the online setting. If the one encoded legitimate or unauthorized device is classified as an outlier, it is counted as an outlier. This was repeated until all of the test set were classified as an outlier or not. In the study, Sklearn's implementation of Local Outlier Factor was used with a threshold over the negative outlier factor. The normal outlier factor is determined by calculating the density of every point. Density is defined as the distance between the point of interest and its k nearest neighbors. If a point has a much smaller density compared to its neighbors, then it is considered an outlier. The negative outlier factor is the opposite of the regular outlier factor, meaning that a smaller value implies normality because it has a larger density and a larger value implies abnormality because it has a smaller density.

4.4.4 Zero-Shot Learning

The DBSCAN clustering algorithm was used on encoded data of all trained networks due to its ability to form complex-shaped clusters. DBSCAN also does not require knowing the exact number of clusters which fits the use case since one would not know the exact number of unauthorized devices in an online setting. Only encoded unauthorized devices are considered in order to compare how well each network is able to learn general features of RF signals. This corresponds to the typical ZSL setting where only the unknown classes are in the test set. Since DBSCAN is an unsupervised method, a metric is needed in order to compare the performance of the models. Since it is desirable to separate unseen classes into their own clusters, each unseen class will be assigned the cluster that contains a majority of that unseen class. "Clustering accuracy" will then be defined as the percent of each class that makes up their respective majority. For example, if 40 out of 100 signals are clustered into one cluster, and the rest are uniformly spread through several other clusters, then a clustering accuracy of 40% is assigned.

Using sklearn's implementation of DBSCAN, the epsilon parameter was initially set to a minimum of the distance between all centroid pairs of the training data. This incorporates information about the shape and size of the known training clusters in order to better cluster similar, but unseen, test clusters. This minimum distance was scaled between one to five times its original value during testing of clustering accuracy as higher or lower values were not shown to increase accuracy.

4.4.5 Generalized Zero-Shot Learning

In a generalized zero-shot learning (GZSL) scheme, it is considered that the online testing stage contains both unauthorized devices and legitimate devices. This should result in smaller accuracy than obtained in the typical ZSL setting due to the test set containing both seen and unseen classes.

4.5 Results

Figure 4.2 shows the results for LOD using receiver operating characteristic (ROC) curves. ROC curves show the rates of true positives (probability of detection) against the rates of false positives (probability of false alarm) at different threshold settings. This means curves closer to the top-left quadrant are better in the sense of having more true-positives and less false-positives.

In this case, the threshold varied between -1 and -1000 over the negative outlier factor in LOD. As shown, the classifiers CNN and MLP outperformed the AE where the CNN scored highest overall. To make this concrete, x-y location $(0.1, 0.5)$ in Figure 4.2 represents a five-to-one ratio of five correctly classified unauthorized devices for every one legitimate device incorrectly classified as unauthorized. During training, the CNN scored an average of 91.52% while MLP scored 90.41% implying higher training accuracy may have an affect on outlier detection.

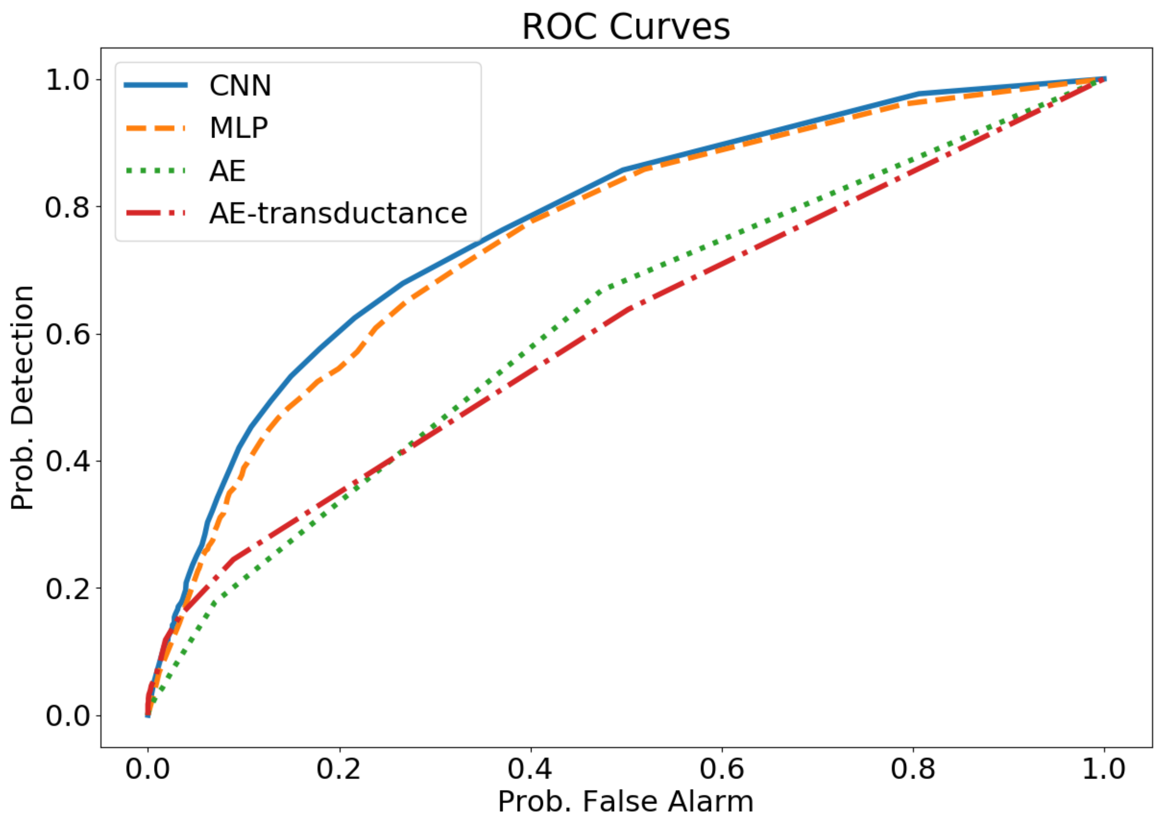
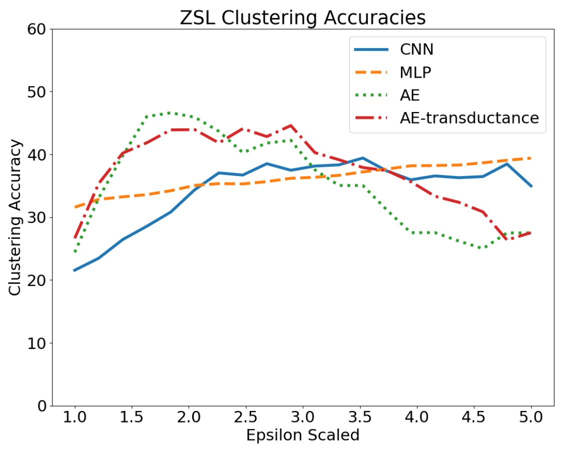


Figure 4.2: ROC curves for the selected models.

Experiments were conducted to evaluate the clustering performance of grouping devices during the testing stage for both ZSL and GZSL settings. Since the CNN performed best in outlier detection, those outliers were used as inputs for clustering in DBSCAN. In particular, the clustering accuracy is reported in terms of the selection of the hyperparameter in DBSCAN clustering algorithm, i.e., epsilon, in Figure 4.3. As shown, the AE outperforms the other models for both ZSL and GZSL settings. Specifically, for ZSL, this clustering accuracy implies AE is able to correctly group 46% of an unknown class on average. Meanwhile, Figure 4.3 also shows both AE's outperforming the CNN, and the AE-transductance providing no improvement over the AE.



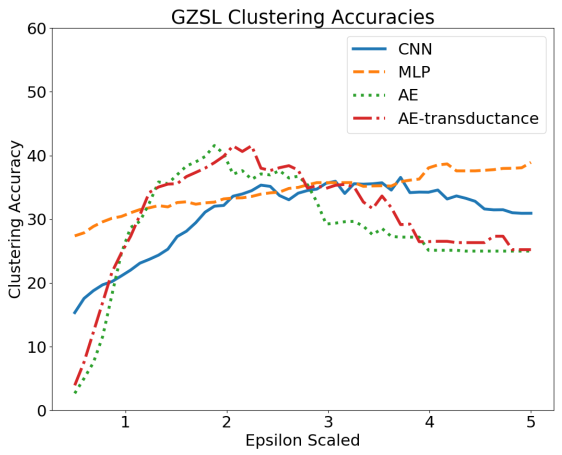


Figure 4.3: Clustering accuracies for both the ZSL and GZSL settings. As expected, clustering in the ZSL setting performed marginally better than the GZSL due to not containing legitimate devices in the mix. Both autoencoders performed better than the classification methods.

4.6 Conclusions and Future Work

ZSL and GZSL for wireless RF-signals were successfully implemented and validated, both of which have not been attempted before in the literature. For the purpose of gathering enough unauthorized RF-signals in order to classify them, an average clustering accuracy of 42% in the GZSL setting was achieved. Whether this is sufficient for outlier detection depends on how many RF-signals are required and how fault-tolerant the proposed model is to mis-clustered data.

Outlier detection performed best to identify unauthorized devices using features learned by CNN-based classifiers, achieving a five-to-one legitimate-to-unauthorized device ratio. This ratio tentatively seemed to improve with greater model accuracy implying that models that achieve closer to 100% accuracy may show performance improvements in outlier detection; however, this may only be the case if those models avoid overconfidence in the face of novel classes.

For clustering in both typical ZSL and GZSL settings, it appears that normal classification models learn features that discriminate seen classes but do not generalize to previously unseen classes. Autoencoders show more promise and may be able to learn features that do generalize, and larger datasets may further improve both autoencoders implemented. Additionally, supervised autoencoders may be able to better incorporate labeled data for increased performance in both outlier detection and clustering accuracy.

CHAPTER V

CONCLUSIONS

This work presents state-of-the-art results for classification of wireless networks for both complex and real-valued networks. This substantiates the claim that wireless physical fingerprints are a viable metric for differentiating legitimate and unauthorized wireless devices. Additional improvements were also found with preprocessing using the FFT.

Both ZSL and GZSL obtained non-trivial results in the novel application of wireless physical fingerprints. The combination of outlier detection and clustering allowed a separation between legitimate and unauthorized devices and further separation of unauthorized devices into their own clusters. Specifically, a 42% average clustering accuracy was achieved in the GZSL setting; this may be sufficient for further purposes such as localization depending on how many samples of the unauthorized devices were captured and how many are required for localization. Higher clustering accuracies would lower the amount of unauthorized device samples required to be captured.

5.1 Further Research

The dataset used included only nine devices with approximately 100 samples each. Gathering a larger dataset in both device number and samples captured would be beneficial for testing the generalization of the methods presented. A larger dataset on the order of 1000 samples per device may benefit these DNNs. Extending the complex NN to accept complex-valued inputs may also improve performance. The data could be additionally be preproccessed by the short time fourier transform and the wavelet transform instead of the FFT.

For both the ZSL and GZSL settings, classification models that obtain higher accuracies than those presented in this work may learn features that generalize to previously unseen classes. Using state-of-the-art outlier detection methods would also increase performance in the outlier detection portion of ZSL and may also lead to enhanced feature selection. In this work, AEs were used for feature selection, but they were unsupervised models. Modified AEs that allow usage of labeled data may achieve greater clustering accuracies.

REFERENCES

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems,", 2015, Software available from tensorflow.org.

[2] M. A. Al-Ammar, S. Alhadhrami, A. Al-Salman, A. Alarifi, H. S. Al-Khalifa, A. Alnafessah, and M. Alsaleh, "Comparative survey of indoor positioning technologies, techniques, and algorithms," *2014 International Conference on Cyberworlds*. IEEE, 2014, pp. 245–252.

[3] Anaconda, "Anaconda Software Distribution. Computer software. Vers. 2-2.4.0. Anaconda,", https://anaconda.com, 2016.

[4] P. Bahl and V. N. Padmanabhan, "RADAR: An in-building RF-based user location and tracking system," *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064)*. IEEE, 2000, vol. 2, pp. 775–784.

[5] G. Baldini and G. Steri, "A survey of techniques for the identification of mobile phones using the physical fingerprints of the built-in components," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, 2017, pp. 1761–1789.

[6] J. E. Ball, D. T. Anderson, and C. S. Chan, "Comprehensive survey of deep learning in remote sensing: theories, tools, and challenges for the community," *Journal of Applied Remote Sensing*, vol. 11, no. 4, 2017, pp. 042609–1 – 042609–54.

[7] C. Basri and A. El Khadimi, "Survey on indoor localization system and recent advances of WIFI fingerprinting technique," *2016 5th International Conference on Multimedia Computing and Systems (ICMCS)*. IEEE, 2016, pp. 253–259.

[8] B. Bloessl, M. Segata, C. Sommer, and F. Dressler, "Performance Assessment of IEEE 802.11p with an Open Source SDR-based Prototype," *IEEE Transactions on Mobile Computing*, vol. 17, no. 5, May 2018, pp. 1162–1175.

[9] V. Brik, S. Banerjee, M. Gruteser, and S. Oh, "Wireless device identification with radiometric signatures," *Proceedings of the 14th ACM international conference on Mobile computing and networking*, 2008, pp. 116–127.

[10] T. J. Carbino, M. A. Temple, and J. Lopez, "A comparison of PHY-based fingerprinting methods used to enhance network access control," *IFIP International Information Security and Privacy Conference*. Springer, 2015, pp. 204–217.

[11] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, 2011, pp. 27:1–27:27, Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[12] F. Chen, Q. Yan, C. Shahriar, C. Lu, W. Lou, and T. C. Clancy, "On passive wireless device fingerprinting using infinite hidden markov random field," *submitted for publication*, 2017.

[13] F. Chollet, "keras,", https://github.com/fchollet/keras, 2015.

[14] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs),", 2015.

[15] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems (MCSS)*, vol. 2, no. 4, 1989, pp. 303–314.

[16] B. Danev, D. Zanetti, and S. Capkun, "On physical-layer identification of wireless devices," *ACM Computing Surveys (CSUR)*, vol. 45, no. 1, 2012, pp. 1–29.

[17] J. S. Dramsch and Contributors, "Complex-Valued Neural Networks in Keras with Tensorflow,", 2019.

[18] M. Elhoseiny, B. Saleh, and A. Elgammal, "Write a classifier: Zero-shot learning using purely textual descriptions," *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 2584–2591.

[19] GNU Radio Website, ,", accessed April 2020.

[20] M. Hahsler, M. Piekenbrock, and D. Doran, "dbscan: Fast Density-Based Clustering with R," *Journal of Statistical Software*, vol. 91, no. 1, 2019, pp. 1–30.

[21] S. He and S.-H. G. Chan, "Wi-Fi fingerprint-based indoor positioning: Recent advances and comparisons," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, 2015, pp. 466–490.

[22] V. Honkavirta, T. Perala, S. Ali-Loytty, and R. Piché, "A comparative survey of WLAN location fingerprinting methods," *2009 6th workshop on positioning, navigation and communication*. IEEE, 2009, pp. 243–251.

[23] J. Jang and S. Hong, "Indoor Localization with WiFi Fingerprinting Using Convolutional Neural Network," *2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN)*, 2018, pp. 753–758.

[24] R. Khullar and Z. Dong, "Indoor localization framework with WiFi fingerprinting," *2017 26th Wireless and Optical Communication Conference (WOCC)*, 2017, pp. 1–6.

[25] H. J. Kim and A. Y. Ng, "Stable adaptive control with online learning," *Advances in Neural Information Processing Systems*, 2005, pp. 977–984.

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[27] R. Klein, M. A. Temple, M. J. Mendenhall, and D. R. Reising, "Sensitivity analysis of burst detection and RF fingerprinting classification performance," *2009 IEEE International Conference on Communications*. IEEE, 2009, pp. 1–5.

[28] R. W. Klein, M. A. Temple, and M. J. Mendenhall, "Application of wavelet-based RF fingerprinting to enhance wireless network security," *Journal of Communications and Networks*, vol. 11, no. 6, 2009, pp. 544–555.

[29] W. A. Knaus, E. A. Draper, D. P. Wagner, and J. E. Zimmerman, "APACHE II: a severity of disease classification system.," *Critical care medicine*, vol. 13, no. 10, 1985, pp. 818–829.

[30] T. Kohno, A. Broido, and K. C. Claffy, "Remote physical device fingerprinting," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 2, 2005, pp. 93–108.

[31] S. Kordumova, T. Mensink, and C. G. Snoek, "Pooling objects for recognizing scenes without examples," *Proceedings of the 2016 ACM on international conference on multimedia retrieval*, 2016, pp. 143–150.

[32] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012, pp. 1097–1105.

[33] F. Lanze, A. Panchenko, B. Braatz, and A. Zinnen, "Clock skew based remote device fingerprinting demystified," *2012 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2012, pp. 813–819.

[34] X. Li, Y. Guo, and D. Schuurmans, "Semi-supervised zero-shot classification with label representation learning," *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 4211–4219.

[35] Y. Li, D. Wang, H. Hu, Y. Lin, and Y. Zhuang, "Zero-shot recognition using dual visual-semantic mapping paths," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 3279–3287.

[36] P. J. Lisboa, "Industrial use of safety-related artificial neural networks," *HSE CR 327/2001*, 2001.

[37] H. Liu, H. Darabi, P. Banerjee, and J. Liu, "Survey of wireless indoor positioning techniques and systems," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 37, no. 6, 2007, pp. 1067–1080.

[38] K. Merchant, S. Revay, G. Stantchev, and B. Nousain, "Deep learning for RF device fingerprinting in cognitive communication networks," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, 2018, pp. 160–167.

[39] G. Minaev, A. Visa, and R. Piché, "Comprehensive survey of similarity measures for ranked based location fingerprinting algorithm," *2017 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. IEEE, 2017, pp. 1–4.

[40] M. Nowicki and J. Wietrzykowski, "Low-effort place recognition with WiFi fingerprints using deep learning," *International Conference Automation*. Springer, 2017, pp. 575–584.

[41] J. Padilla, P. Padilla, J. Valenzuela-Valdés, J. Ramírez, and J. Górriz, "RF fingerprint measurements for the identification of devices in wireless communication networks based on feature reduction and subspace transformation," *Measurement*, vol. 58, 2014, pp. 468–475.

[42] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., "Scikit-learn: Machine learning in Python," *Journal of machine learning research*, vol. 12, no. Oct, 2011, pp. 2825–2830.

[43] P. J. Phillips and P. J. Phillips, "Support vector machines applied to face recognition," 1998.

[44] S. U. Rehman, K. Sowerby, and C. Coghill, "Analysis of receiver front end on the performance of RF fingerprinting," *2012 IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications-(PIMRC)*. IEEE, 2012, pp. 2494–2499.

[45] S. U. Rehman, K. Sowerby, and C. Coghill, "RF fingerprint extraction from the energy envelope of an instantaneous transient signal," *2012 Australian Communications Theory Workshop (AusCTW)*. IEEE, 2012, pp. 90–95.

[46] S. U. Rehman, K. W. Sowerby, S. Alam, and I. Ardekani, "Radio frequency fingerprinting and its challenges," *2014 IEEE Conference on Communications and Network Security*. IEEE, 2014, pp. 496–497.

[47] M. Rohrbach, M. Stark, G. Szarvas, I. Gurevych, and B. Schiele, "What helps where–and why? semantic relatedness for knowledge transfer," *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, 2010, pp. 910–917.

[48] W. C. Suski II, M. A. Temple, M. J. Mendenhall, and R. F. Mills, "Using spectral fingerprints to improve wireless network security," *IEEE GLOBECOM 2008-2008 IEEE Global Telecommunications Conference*. IEEE, 2008, pp. 1–5.

[49] L. Tarassenko, "LogiCook and QUESTAR: two case studies in successful technology transfer," 1997.

[50] R. L. Thorndike, "Who belongs in the family?," *Psychometrika*, vol. 18, no. 4, 1953, pp. 267–276.

[51] J. Torres-Sospedra, R. Montoliu, A. Martínez-Usó, J. P. Avariento, T. J. Arnau, M. Benedito-Bordonau, and J. Huerta, "UJIIndoorLoc: A new multi-building and multi-floor database for WLAN fingerprint-based indoor localization problems," *2014 international conference on indoor positioning and indoor navigation (IPIN)*. IEEE, 2014, pp. 261–270.

[52] C. Trabelsi, O. Bilaniuk, Y. Zhang, D. Serdyuk, S. Subramanian, J. F. Santos, S. Mehri, N. Rostamzadeh, Y. Bengio, and C. J. Pal, "Deep complex networks," *arXiv preprint arXiv:1705.09792*, 2017.

[53] O. Ureten and N. Serinken, "Bayesian detection of Wi-Fi transmitter RF fingerprints," *Electronics Letters*, vol. 41, no. 6, 2005, pp. 373–374.

[54] Q. D. Vo and P. De, "A survey of fingerprint-based outdoor localization," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, 2015, pp. 491–506.

[55] W. Wang, V. W. Zheng, H. Yu, and C. Miao, "A survey of zero-shot learning: Settings, methods, and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, 2019, pp. 1–37.

[56] M. Wolter and A. Yao, "Complex gated recurrent neural networks," *Advances in Neural Information Processing Systems*, 2018, pp. 10536–10546.

[57] Q. Xu, R. Zheng, W. Saad, and Z. Han, "Device fingerprinting in wireless networks: Challenges and opportunities," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, 2015, pp. 94–104.

[58] X. Xu, F. Shen, Y. Yang, J. Shao, and Z. Huang, "Transductive visual-semantic embedding for zero-shot learning," *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*, 2017, pp. 41–49.

[59] F. Zafari, A. Gkelias, and K. K. Leung, "A survey of indoor localization systems and technologies," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, 2019, pp. 2568–2599.

[60] Z. Zhang and V. Saligrama, "Zero-shot learning via joint latent similarity embedding," *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 6034–6042.

APPENDIX A

KERAS NETWORK CODES

A.1 Import commands

```
    from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

# Tell python to find the complexnn directory
import sys
sys.path.append(r“path to complexnn”) # Put your path here

import complexnn
import numpy as np
import keras
import tensorflow as tf
from keras import models, optimizers, regularizers
from keras.models import Sequential
from keras.layers import Activation, Dropout, Dense, Flatten, Conv1D, Input, Layer
from keras.layers import MaxPooling1D
from keras.models import Model
from keras.utils import to_categorical
from tensorflow.python.keras import backend as K
from keras.metrics import categorical_accuracy
from datetime import datetime
from packaging import version
from math import pi
from sklearn.model_selection import train_test_split
from sklearn import svm
```

A.2 Keras code for DNN-R1

```
    input_shape = [data_vector_size]
model = Sequential()
model.add(Dense(800, activation='relu', input_shape=input_shape))
model.add(Dense(200, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dropout(rate=0.50))
model.add(Dense(num_classes, activation='softmax'))
```

A.3 Keras code for DNN-C1

```
    input_shape = [data_vector_size]
inputs = Input(shape=input_shape)
z = inputs
z = complexnn.dense.ComplexDense(800, activation=ModReLU, input_dim=data_vector_size)(z)
z = complexnn.dense.ComplexDense(200, activation=ModReLU)(z)
z = complexnn.dense.ComplexDense(50, activation=ModReLU)(z)
```

```
za = complexnn.utils.GetAbs()(z)

za = tf.cast(za, dtype=tf.float32)

zd = Dropout(rate=0.60)(za)

outputs = Dense(num_classes, activation='softmax')(zd)

model = Model(inputs, outputs)
```

A.4 Keras code for CNN-R1

```
    input_shape = (data_vector_size, 1)

model = Sequential()

model.add(Conv1D(32, 12, activation='relu', padding='same', strides=2, input_shape=input_shape))

model.add(MaxPooling1D(2))

model.add(Conv1D(64, 3, activation='relu', padding='valid', strides=2))

model.add(MaxPooling1D(2))

model.add(MaxPooling1D(2))

model.add(Dropout(rate=0.50))

model.add(Flatten())

model.add(Dense(32, activation='relu'))

model.add(Dense(32, activation='relu'))

model.add(Dropout(rate=0.50))

model.add(Dense(num_classes, activation='softmax'))
```

A.5 Keras code for CNN-C1

```
      input_shape = (data_vector_size, 1)
inputs = Input(shape=input_shape)
z = inputs
# Had to use regular Conv1D instead of ComplexConv1D for first layer to get network to work
z = (Conv1D(32, 12, activation=ModReLU, padding='same', strides=2, input_shape=input_shape))(z)
z = (MaxPooling1D(2))(z)
z = (complexnn.conv.ComplexConv1D(64, 3, padding='valid', activation=ModReLU, strides=2,
      kernel_initializer='complex_independent'))(z)
z = MaxPooling1D(2)(z)
z = MaxPooling1D(2)(z)
z = Dropout(rate=0.50)(z)
z = Flatten()(z)
z = complexnn.dense.ComplexDense(32, activation=ModReLU)(z)
z = complexnn.dense.ComplexDense(32, activation=ModReLU)(z)
z = Dropout(rate=0.50)(z)
outputs = Dense(num_classes, activation='softmax')(z)
model = Model(inputs, outputs)
```

A.6 ModReLU activation function

```
# ModReLU(z) = ReLU(|z| + b) * z / |z|

def ModReLU(z, b=0.01):

    absz = K.abs(z) + 1.0e-12

    out = K.relu(absz + b) * z / absz

    return out
```

A.7 Keras code for CNN

```
input_shape = [data_vector_size]

model = Sequential()

model.add(Conv1D(16, kernel_size=(5), activation='relu', input_shape=input_shape))

model.add(Conv1D(64, kernel_size=(1), activation='relu'))

model.add(MaxPooling1D(pool_size=(1)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(32, activation='relu'))

model.add(Dropout(rate=0.50))

model.add(Dense(num_classes, activation='softmax'))
```

A.8 Keras code for MLP

```
input_shape = Input(shape=(data_vector_size,))

l1 = Dense(50, activation='relu')(input_shape)

l1 = Dropout(rate=0.3)(l1)

l1 = Dense(50, activation='relu')(l1)

l1 = Dropout(rate=0.3)(l1)

l1 = Dense(50, activation='relu')(l1)

l2 = Dense(num_classes, activation='softmax')(l1)

model = Model(input_shape, l2)
```

A.9 Keras code for AE

```
input_shape = Input(shape=(data_vector_size,))

encoded = Dense(200)(input_shape)

encoded = Dense(100)(encoded)

encoded = Dense(40)(encoded)

encoded = Dense(13)(encoded)

decoded = Dense(40)(encoded)

decoded = Dense(100)(decoded)

decoded = Dense(200)(decoded)

decoded = Dense(data_vector_size)(decoded)

autoencoder = Model(input_shape, decoded)
```

```
encoder = Model(input_shape, encoded)
```

A.10 Anaconda installation code

```
conda create –name complex python=3.7.1 numpy=1.16.2

conda activate complex

conda install -c conda-forge packaging

conda install -c anaconda scikit-learn

conda install -c conda-forge keras
```

cd to code directory

```
pip install git

git clone https://github.com/JesperDramsch/keras-complex

cd keras_complex

pip install -r requirements-nogpu.txt

pip install keras-complex
```